Learning Algorithms for the Control of Routing in Integrated Service Communication Networks

Jonathan Mark Reeve

School of Engineering

University of Durham

September 1998

A thesis submitted for the degree of Doctor of Philosophy (Ph.D.)
of the University of Durham.

Jonathan Mark Reeve

Learning Algorithms for the Control of Routing in Integrated Service Communication Networks

Ph.D. 1998.

Abstract

There is a high degree of uncertainty regarding the nature of traffic on future integrated service networks. This uncertainty motivates the use of adaptive resource allocation policies that can take advantage of the statistical fluctuations in the traffic demands. The adaptive control mechanisms must be 'lightweight', in terms of their overheads, and scale to potentially large networks with many traffic flows. Adaptive routing is one form of adaptive resource allocation, and this thesis considers the application of Stochastic Learning Automata (SLA) for distributed, lightweight adaptive routing in future integrated service communication networks. The thesis begins with a broad critical review of the use of Artificial Intelligence (AI) techniques applied to the control of communication networks. Detailed simulation models of integrated service networks are then constructed, and learning automata based routing is compared with traditional techniques on large scale networks.

Learning automata are examined for the 'Quality-of-Service' (QoS) routing problem in realistic network topologies, where flows may be routed in the network subject to multiple QoS metrics, such as bandwidth and delay. It is found that learning automata based routing gives considerable blocking probability improvements over shortest path routing, despite only using local connectivity information and a simple probabilistic updating strategy. Furthermore, automata are considered for routing in more complex environments spanning issues such as multi-rate traffic, trunk reservation, routing over multiple domains, routing in high bandwidth-delay product networks and the use of learning automata as a background learning process.

Automata are also examined for routing of both 'real-time' and 'non-real-time' traffics in an integrated traffic environment, where the non-real-time traffic has access to the bandwidth 'left over' by the real-time traffic. It is found that adopting learning automata for the routing of the real-time traffic may improve the performance to both real and non-real-time traffics under certain conditions. In addition, it is found that one set of learning automata may route both traffic types satisfactorily.

Automata are considered for the routing of multicast connections in receiver-oriented, dynamic environments, where receivers may join and leave the multicast sessions dynamically. Automata are shown to be able to minimise the average delay or the total cost of the resulting trees using the appropriate feedback from the environment. Automata provide a distributed solution to the dynamic multicast problem, requiring purely local connectivity information and a simple updating strategy.

Finally, automata are considered for the routing of multicast connections that require QoS guarantees, again in receiver-oriented dynamic environments. It is found that the distributed application of learning automata leads to considerably lower blocking probabilities than a shortest path tree approach, due to a combination of load balancing and minimum cost behaviour.

Declaration

I hereby declare that this thesis is a record of work undertaken by myself, that it has not been the subject of any previous application for a degree, and that all sources of information have been duly acknowledged.

Acknowledgements

I would like to express my thanks to my supervisor, Professor Phil Mars, for his guidance and encouragement throughout my stay at Durham.

I would also like to acknowledge an EPSRC 'CASE' award from BT labs. Thanks to my supervisor at BT, Dr. Terry Hodgkinson, who made valuable contributions to the project and helped to make my stays at BT labs run smoothly. Thanks also to Martin Tatham and Graham Brown at BT labs for their useful discussions.

Thanks to the guys in the lab who have helped me out over the years. In particular, thanks must go to Martin, Steve, Jason, Mark, Phil, Fan and Fred.

Finally, thanks to my parents for supporting me through all these years at Durham.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Networks
AS	Autonomous System
ASAP	As Soon As Possible
ATM	Asynchronous Transfer Mode
BGP	Border Gateway Protocol
CAC	Connection Admission Control
CBT	Core/centre Based Trees
CBQ	Class-based-queuing
CLR	Cell Loss Ratio
CST	Constrained Steiner Tree
DAR	Dynamic Alternative Routing
DNHR	Dynamic Non-hierarchical Routing
DVRMP	Distance Vector Multicast Routing Protocol
FAM	Fuzzy Associative Memory
FIFO	First-in-first-out
FTP	File Transfer Protocol
ICMP	Internet Control Message Protocol
IDMR	Inter-domain Multicast Routing
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISPN	Integrated Services Packet Network
KMB	Kou-Markowsky-Berman
LAN	Local Area Network
LRI	Linear Reward Inaction
LRP	Linear Reward Penalty
MOSPF	Multicast Extensions to OSPF
MST	Minimum Steiner Tree
NRT	Non-real-time
OSPF	Open Shortest Path First
pdf	Probability Density Function
PIM	Protocol Independent Multicast

PIM-DM	Protocol Independent Multicast – Dense Mode
PIM-SM	Protocol Independent Multicast – Sparse Mode
PNNI	Private Network-Network Interface
POTS	Plain Old Telephone Service
QoR	Quality-of-Route
QoS	Quality-of-Service
QOSPF	QoS Extensions to OSPF
RIP	Routing Information Protocol
RP	Rendezvous Point
RSVP	Resource Reservation Protocol
RT	Real-time
SDH	Synchronous Digital Hierarchy
SDR	Source Demand Routing
SLA	Stochastic Learning Automata/Automaton
SPT	Shortest Path Trees
TCP	Transmission Control Protocol
ToS	Type-of-Service
TTL	Time-to-live
VBR	Variable Bit Rate
WFQ	Weighted Fair Queuing
WWW	World Wide Web

Chapter 1

Introduction

1.1. Future Networks

Future networks are likely to contain considerable functionality not contained in today's networks. Example functions include the need for multiple levels of Quality-of-Service (QoS), multipoint or multicast communication and mobile networking issues. There is a need then to consider enhanced control mechanisms that may deal with this additional complexity whilst ensuring that the control mechanisms themselves are simple enough to be implemented in a practical network. To take advantage of economies of scale, it is desirable to integrate multiple services onto one network infrastructure. Such networks have been termed, 'integrated-services networks'. Integrating multiple traffic types onto one network also creates further complexity since the network control mechanisms must be able to scale to potentially very large networks with many different services and traffic flows. Another characteristic of integrated-services networks is that we can expect considerable uncertainty regarding the traffic flows on these networks. From the 'central limit theorem', we expect the aggregate traffic from all services to approach a Gaussian distribution. However, the variance of the aggregate distribution is proportional to the variance of the individual distributions. The variability of each individual traffic type or service means that the aggregate usage will be more variable [1]. This aggregate may be even more unpredictable in that a significant proportion of the traffic may involve computer (rather than human) communications and we expect this type of communication to be much less predictable than human communication behaviour [1].

Due to this increasing uncertainty and the fact that efficient control schemes are unlikely to be able to be produced by design, some form of adaptive control becomes attractive to perform adaptive resource allocation in the network to take advantage of the statistical fluctuations in the use of the network. Given the complexity of such systems however, mathematical models necessary for adaptive control can rarely be constructed. This has motivated the study of Artificial Intelligence (AI) techniques for performing adaptive control in networks.

Adaptive routing can be considered as one form of adaptive resource allocation, where a routing algorithm can improve overall throughputs by routing flows along the proper links which form a route. In this thesis, the aim is to examine the potential of learning algorithms for the adaptive routing of the various traffic flows that will make up an integrated-services network. Integrated-services networks

introduce new problems over previous circuit and packet switched networks, including the routing of flows based on multiple (QoS) metrics, the incorporation of multi-rate traffic, aggregated routing and very large bandwidth-delay products. Additionally, there is a need to consider the routing of real-time and non-real-time traffic elements and how these may interact. For example, if we choose to operate an adaptive routing algorithm for the real-time traffic element, we may be able to adaptively route the non-real-time traffic at very little incremental cost. Finally, there is a need to consider the adaptive routing of multicast as well as unicast connections, since there is a widespread agreement that multicast will be an important technology for conserving bandwidth and signalling overhead in future networks. In this thesis, we have applied learning algorithms to the routing of multicast connections and this is the first time that this has been attempted in the literature to the best of our knowledge.

1.2. Network Control in Integrated Service Networks

In this chapter, we go on to describe the fundamental control problems present in integrated service networks. Firstly, it is necessary to introduce what is meant by integrated service networks, and the basic network architectural components which they are expected to support. We then go on to explain why the control of these networks presents such a challenge, discussing the need for some form of adaptive control and explaining how network control techniques may be compared with one another.

1.3. Integrated Services Networks

Future networks will almost certainly be required to support a wide variety of services. One possible way to achieve this would be to design and build a separate network to support each service. Thus, there would be discrete networks to support voice, video, file transfer etc.... The benefit of this approach is that each network can be individually optimised to the needs of the single service which it must support. POTS (Plain Old Telephone Service) is an example of a network that has been engineered, extremely effectively, to (originally) support the needs of a single service, namely, voice. Designing and implementing a separate network for each service incurs considerable overhead however, since control and management costs must be duplicated for each individual network, thereby neglecting the advantages of possible economies of scale. The support of multiple services on a single network infrastructure also enables the (possibly significant) statistical sharing of resources. For these reasons, Integrated Service Networks are deemed a worthy goal, although there are considerable political, administrative and technical problems to solve before their implementation becomes a reality.

Current network infrastructures generally only support a single 'Quality-of-Service' (QoS). Examples that will be used throughout this thesis are the current telephony network and the Internet. The telephony network has grown from the early days of direct point-to-point links to a sophisticated switched digital network. Nevertheless, the current telephony network only supports a single quality-of-service, based on building blocks of 64kbit/s circuit allocations used for services such as voice, fax and modems. A call

submitted to the telephony network is generally admitted with a high probability, the ensuing end-to-end delay being low, as bounded by design. In this thesis, we invariably refer to traffic with bounded delays as 'real-time' (RT) traffic.

The Internet has grown out of the original research into the ARPANET [2] during the 1960s, and was originally designed with military applications in mind, the main aim being to create a resilient packet switched network that could forward critical data from one point to another despite many destroyed links and/or nodes. The Internet currently only supports a very simple quality-of-service, where no assurances are given about when or even if data packets arrive at a destination. In this thesis, we will refer to traffic which requires no explicit delay bound as 'non-real-time' (NRT) or 'best-effort' traffic. Although the above distinction between RT and NRT traffic is rather coarse, it is sufficient for the experiments in this thesis.

Designing a network for multiple services confronts the network designer with inevitable trade-offs, since services may have vastly differing traffic characteristics. Since it is impossible to optimise on all fronts, we seek an architecture that will provide a practical compromise, whilst retaining flexibility for possible future service implementations. The main proponents of the telephony network and the Internet, looking to expand the range of services that may be supported by their networks, have proposed architectures to support integrated services. The telephony based companies and others have proposed ATM, the 'Asynchronous Transfer Mode', and formed a consortium of companies known as the 'ATM forum'. The IETF (Internet Engineering Task Force) have a working group which has proposed the 'Int-serv' and more recent 'Differential Services' models, which propose a number of extensions to the Internet's best-effort service model in order to support real-time applications. The aim of both these groups is essentially the same, to support a range of applications with differing QoS requirements on a single network infrastructure. A simple example would be to support both high quality voice and traditional data services on a solitary network. Comprehensive arguments for extending the basic service model of the Internet, and for Integrated Service networks in general are presented in [1].

1.4. Integrated-Services Architecture

In this section, we describe the basic architectural considerations necessary to support Integrated Services. There exists a large body of literature concerned with providing real-time service in a packet switched network, much of the work concentrating on scheduling algorithms, admission control, reservation protocols and flow specifications (see [3], [4], [5], [6], [7], [8], [9], [10], [11]). In [4], four basic architectural aspects are deemed necessary to define an Integrated Services Packet Network (ISPN) architecture. These are the Service Model, Packet Scheduling, Service Interface and Connection Admission Control (CAC) elements. We comment briefly about each of these aspects in turn.

1.4.1. The Service Model

The Service Model represents the most important single element in defining a ISPN since it is the most enduring part of a network architecture [4]. Although the underlying network technology and overlying suite of applications may evolve, the need for compatibility requires that the existing service model remain largely unchanged. The design of a Service Model is in turn driven by speculation over the requirements of applications and users, both present and future. It is fundamental that as much as possible, the service model be designed that avoids assumptions about the type of traffic using it. One example service model which has been proposed is presented in Figure 1.1 below, and is based on a service model presented in [4].

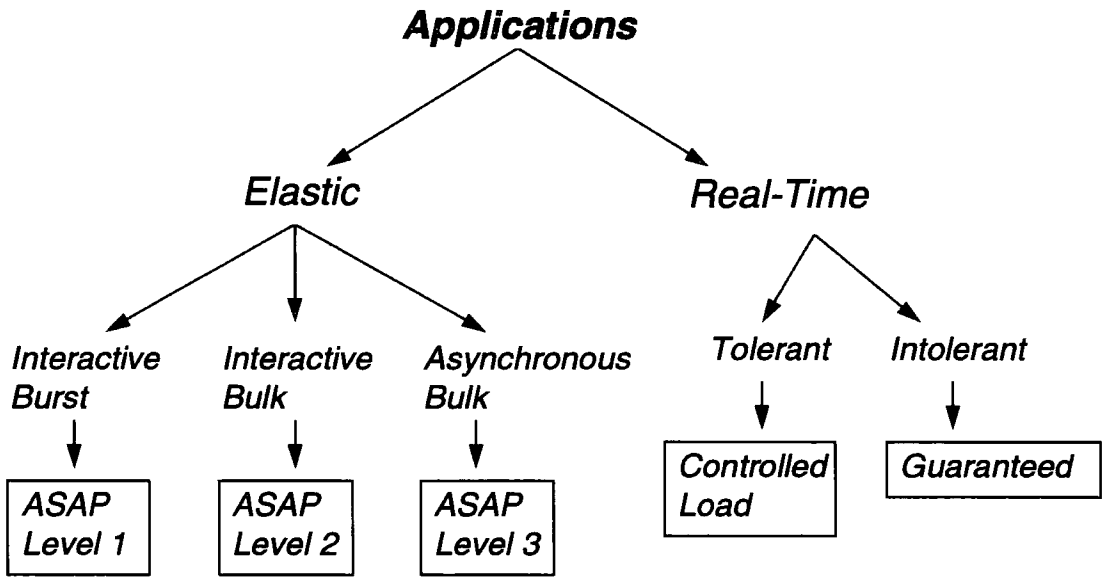


Figure 1.1. – A proposed service model

The above service model is derived upon consideration of packet delay as the service commitment, since delay is generally considered to be the most central quality-of-service. At the top level of the tree, applications can be grouped into two broad classes. Real-time (RT) applications are defined as those applications that require the data in each packet by a certain time and, if the packet arrives after this time, the data is essentially useless to the application. In contrast, 'elastic' or non-real-time (NRT) applications are defined as those applications that will use the data in the packets, no matter how 'late' these arrive. Performance to the elastic applications will increase with decreased packet delay however, which will in turn increase user satisfaction with the elastic service. This may be important since there is widespread agreement within the industry that demand for these elastic services will be quite large [12]. Within the two broad classes, there are finer divisions representative of typical NRT and RT applications respectively. The acronym 'ASAP' means, 'As Soon As Possible', and is used in [4], since the term 'best-effort' is synonymous with FIFO (first-in-first-out) queues which may not necessarily be used in the future Internet. For the elastic applications, three finer application classes are arbitrarily defined, and

reflect their relative delay sensitivities. Part of the problem from a network control perspective then is, 'what granularity of the service model should we design controls for?'. In this case, does having discrete controls for fine classes of NRT traffics improve the network performance sufficiently to justify the cost of their implementation? The 'Int-Serv' working group has proposed three basic services, these being traditional best-effort, 'controlled load' [13] and 'guaranteed service' [14]. It is envisioned that routers would maintain a separate queue for each service class and some form of priority scheduling serving guaranteed, controlled load and best-effort in that order. The guaranteed class is designed for those RT applications that need a perfectly reliable upper bound on the delay of each packet. This bound is calculated assuming worst case behaviour from all other flows in the network. An appropriate Resource Reservation Protocol (e.g. RSVP (Resource Reservation Protocol) [7]), is then used to reserve the necessary resources to provide this worst case delay bound. In [4], it is speculated that the majority of RT applications will be able to tolerate (i.e. adapt to) some late and/or lost packets, and discuss the concept of the controlled/predictive load service, which would be designed to behave like a, 'lightly loaded Internet', providing RT flows with 'fairly' reliable delay bounds. The idea behind the controlled load service is to increase network efficiency, since much higher network utilisations are attainable when one relaxes the service requirements from perfectly rigid to fairly flexible ones. Again, we are faced with a granularity issue from the network control perspective, in that, should we have discrete control mechanisms for these two variants of RT service? The more recent 'differential services' work approaches the problem of integrated services from another angle, that of much longer term resource contracts between the communicating parties. In a recent differential services proposal [15], there are again three basic service types, these being best-effort, 'premium' and 'assured' service. The main difference between this proposal and the integrated services work is that the admission and set-up procedures for RT traffics are statically configured by design, although it is speculated that these may evolve into dynamic set-up procedures as experience is gained with the architecture. Also, reserving resources on a 'per-flow' basis is unlikely to scale to the wide-area so that mechanisms to reserve resources for groups of flows need to be considered.

1.4.2. Packet Scheduling

Having defined a service model such as the example used above, the network must implement a packet scheduling algorithm to support this model. As pointed out in [4] :

In fact, the packet scheduling algorithm is the most fundamental way in which the network can allocate resources selectively; the network can also allocate selectively via routing or buffer management algorithms, but neither of these by themselves can support a sufficiently general service model.

In the Internet, nodes currently employ a simple FIFO queuing regime. While this has been satisfactory in the past, the need to isolate both users and different traffic types from one another motivates the consideration of more sophisticated scheduling algorithms. The fundamental issues are those of *isolation*

and *sharing* [16]. Examples of algorithms providing isolation are so called *fair queuing* (e.g. see [6]) techniques, where bandwidth is somehow apportioned in equal shares, and sessions are isolated from one another. It is thought in [3] however, that different service classes can have different requirements from a scheduling algorithm and it is shown that while Weighted Fair Queuing (WFQ) [8, 9] is suitable for providing isolation for guaranteed traffic, FIFO has many desirable sharing properties more suited to the needs of the controlled load service. The important point with packet scheduling is that it dictates the control of the network at the finest timescale. While there is some flexibility in the choice of longer timescale control methods, such as routing and connection admission control (CAC), the scheduling algorithm must be carefully chosen since no higher level control method will be able to correct for a badly chosen scheduling algorithm at the design stage.

1.4.3. Service Interface

A well defined service interface becomes necessary once guaranteed and controlled load services are introduced for the RT traffic. The service interface defines the parameters passed between the source (and possibly receiver) and the network and may include Quality-of-Service parameters and characterisation of the traffic source statistics (e.g. peak rate). In the Internet community, traffic statistics are likely to be specified in terms of the *token bucket* filter. In this model, a source is characterised by two parameters, the sending rate r and the size of the bucket b, which represent some measure of the average and bursty behaviour of the source respectively. User traffic flows will need to be *'policed'* (primarily at the edge of the network) to ensure that these flows are conforming to the original traffic contract. An important result derived by Parekh and Gallager [8, 9], is that if traffic is characterised using the token bucket model, and the router implements a WFQ (Weighted Fair Queuing) scheduling algorithm, then there will be an absolute upper bound on the delay of the traffic. It is envisioned [14], that guaranteed service would be provided using such a mechanism.

1.4.4. Connection Admission Control (CAC)

Connection Admission Control is the decision taken by the network to decide whether a new flow can be admitted to meet the QoS requested by the new flow, whilst ensuring that the QoS guarantees made to previously accepted connections are maintained. CAC is needed since resources are finite, and there will be a limit to the number of service requests that may be accepted, although the exact form of a future CAC mechanism is open to question. Indeed, some question the need for admission control at all, believing that overprovisioning of the network will suffice [1]. In a limited resource environment, the admission control could play an important role in allowing the scheduling algorithms to be effective by keeping the aggregate traffic load down to a level where meeting the service commitments is feasible [4]. Admission Control may also play an important role in enforcing link sharing mechanisms, whereby companies would like sharing between such things as protocol types or applications, with a predetermined policy on how this should be carried out (e.g. see [12]).

1.4.5. Routing in Integrated Services Networks

Assuming that there is a dynamic set-up procedure for RT traffic in future networks and that the routes have not been installed by design, a routing protocol will be required to route both NRT and RT traffics from source(s) to destination(s). Although routing is not strictly a formal part of an integrated services architecture necessary to support real-time service as defined above, it can have a significant effect on the total throughput of a particular network through careful choice of proper links and nodes which form a chosen route between the source(s) and destination(s). In Chapters 3, 4, 5 and 6, we examine the problem of routing in integrated services networks for RT and NRT traffics for unicast and multicast routing modes. Our specific interest is how learning algorithms or some form of non-symbolic 'Artificial Intelligence' (AI) might provide superior performance over traditional routing mechanisms, when there is little network state information available.

1.4.6. A Unified Mechanism

The following is a brief description of how we expect the five elements described above to reserve resources for real-time sessions, ultimately via a dynamic Resource Reservation protocol (e.g. RSVP [7]). Applications will simply send packets into the network, the particular service required marked by a field in the packet header. The application may also be required to characterise its traffic flow using some traffic filter model, as described in the Service Interface section above. For RT services, the first packet may well contain a reservation request containing the application's desired QoS via a traffic descriptor, and will be forwarded along some path, determined by the real-time routing algorithm, from source to destination. Each router along this path will contain a local Admission Control module, and decide whether this new flow can be accepted based on the current status of its output links. If the admission control decision is successful at all nodes along the path, the set-up is successful, and the application may send traffic in accordance with the traffic descriptor and receive the requested quality-of-service. Otherwise, the set-up attempt has failed and another path must be tried or the flow should be rejected. If a reservation request is failed by the network, the traffic from the requesting source may or may not be sent via a lower priority service (e.g. best-effort). In Figure 1.2, we present a model of a node to support integrated services taken from [17].

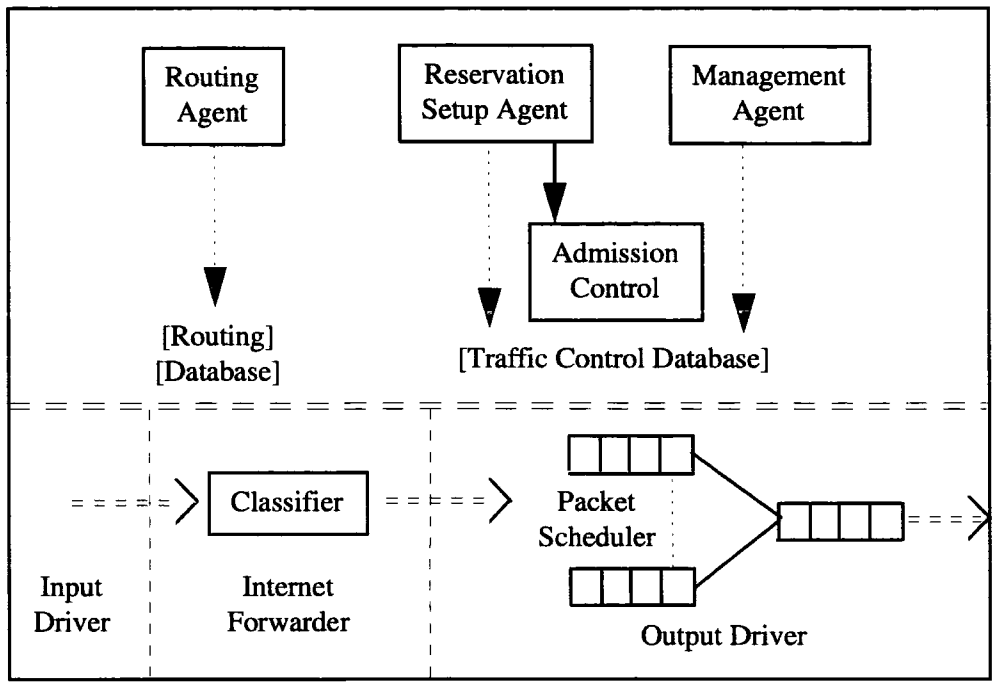


Figure 1.2 - Example of an integrated services node.

1.5. The distributed nature of the Network Control Problem

So far, we have introduced some of the controls likely to be required by an integrated services network architecture. In this section, we explain how network control is a problem in both space and time and why some form of adaptive control is becoming essential. If we consider a communications network under some demand from users, it is the job of the network designer to construct a network which best meets these demands in the most effective and efficient way. Producing an efficient design assumes that we have a good knowledge of the demands likely to be placed on the network. Whilst this has been true of telephony based networks for some time, there is no reason to assume that this situation will continue into the future. The fundamental reason for some form of adaptive control is the sheer uncertainty regarding the nature and volume of these traffic demands on future networks. Once the network is operational, it is the job of network management and control to gather information of these demands and implement the most appropriate control in the network. Network management and control are usually distinguishable through the timescale on which they operate. Network management is traditionally responsible for operational management, maintenance, configuration management, performance management and user administration areas. By network control, we generally mean those lower timescale functions such as flow control, routing or scheduling. If we consider network routing as an example, a network administrator may change routing tables in the network in response to changing traffic demands on a timescale of hours-days and upwards, whilst an automatic control mechanism would be able to operate on timescales significantly less than this.

Centrally controlled networks typically use a 'control node', which gathers state information for the

entire network by sequentially interrogating the nodes of the network. In this way, we build up a world model of the network and may compute the 'optimal' control action, albeit delayed by the interrogation process. The delays inherent with centralised control mean that the information we gather has a higher chance of being 'out-of-date', particularly in dynamic environments. In addition, the database of information stored by the control node may become unrealistically large as the network grows in size. The Internet in part, uses an alternative control technique, where a considerable part of the control process is distributed throughout the nodes of the network. In this way, nodes may make control decisions based on more localised information and will therefore be faster to react, the local information likely to be more accurate (i.e. 'up-to-date') than that gathered from distant nodes. The effect of the superposition of all the local control actions is likely to lead to a sub-optimal control scheme from a global perspective however, since decisions have been made without complete state information. In Table 1.1, we show some of the relative trade-offs of centralised and distributed control.

	Centralised Control	**Decentralised Control**
Computation	High	Low
Communication	High	Low
Database/Storage	Large and Unique	Small and Distributed
Robustness	Low	High
Speed of Response	Slow (since sequential data gathering)	Fast
Optimality of control action	High (since global picture)	Low (since localised picture)

Table 1.1 - Trade-offs of centralised and distributed control

Centralised network architectures designed around stable traffic demands to achieve high utilisations will not be robust to rapid change. Since we expect a high degree of uncertainty regarding future services and traffic mixes, we should look to control mechanisms that make the least assumptions about the traffics using the network. In fact, optimality should perhaps now be defined by how well an architecture may cope with change rather than how well an architecture can optimise its resource usage. We are still interested in making the most of the resources in the network although this aim is secondary to that of flexibility in the face of change. The aim then is to provide a reasonable 'performance' over a wide range of traffic (and topology) conditions rather than 'optimal' performance for a particular instance of traffic statistics (i.e. graceful performance degradation). In summary, we are interested in researching those control mechanisms which enable a greater degree of decentralisation in order to provide a scalable and adaptable network control architecture under uncertain traffic and possibly topological conditions.

Recalling the control mechanisms for integrated service networks, we present a graph showing the relative position of the controls in space and time in Figure 1.3 below. The graph is designed to show the

relative differences between different controls in space and time, rather than any absolute measure. For example, there are routing algorithms that are totally distributed (local information only) and totally centralised (global information). Also, routing decisions may be performed per-packet or on longer timescales such as per-connection. 'Caching' is the process by which information fetched previously is stored in case it is required for future access. For example, information retrieved across a network may be retained in case we require access to it in the future. With 'mirroring', we may choose to locate the same information at multiple sites in the network so that users may fetch information from a local site rather than from a more remote one. For example, video servers may be located at multiple sites in the network to spread the load in the network and increase the network performance.

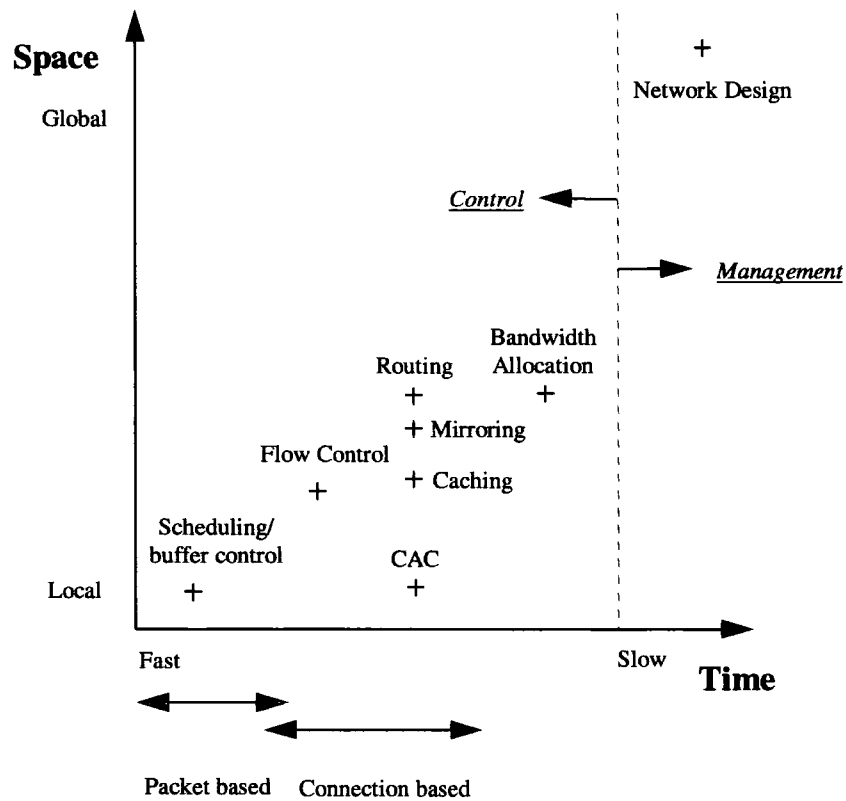


Figure 1.3 - Network control in space/time

1.6. User and Network Control Perspectives

Although we have up till now referred to 'network control', there exists a body of thought that expects control to propagate back to the user as end-user machines become increasingly capable of performing control themselves [18]. An example of this are the differences between source and hop-by-hop routing, which are investigated in more detail later in this thesis. When there is reasonable contention for resources within the network, we should expect users to adapt their behaviour to maximise the resources they see. If the network is also performing adaptation, these network and user control loops may interact if they operate on similar timescales. We show this in Figure 1.4, where the controllers represent the

range of resource control options that are available to the network and user respectively.

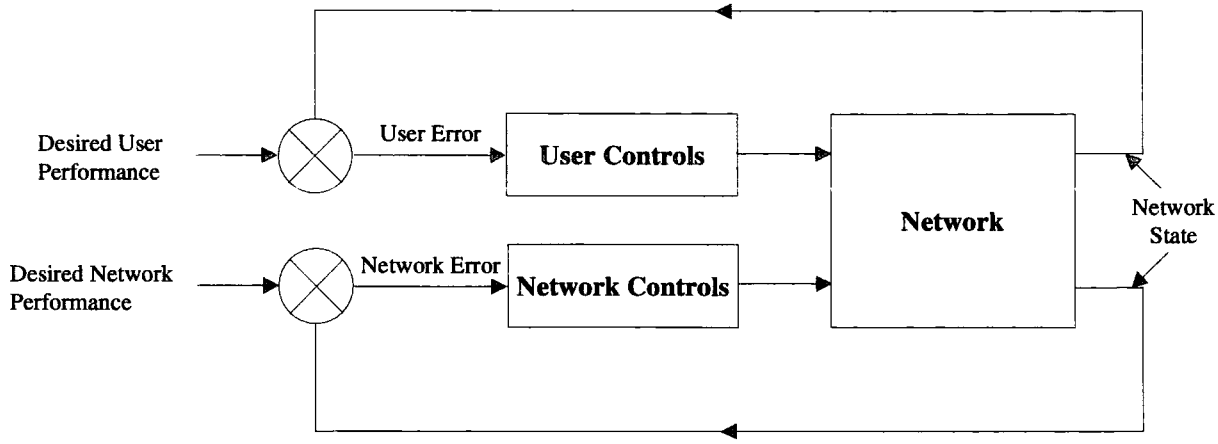


Figure 1.4 - User and network control loops

An example of control loop interaction would occur when we have both hop-by-hop and end-to-end flow controls in operation simultaneously. Hop-by-hop flow control refers to the regulation of the flow between two successive nodes in the network, whereas end-to-end flow control refers to the control of flow between the entry and exit nodes (i.e. source and destination nodes). Consider the case where congestion is detected within the network, and a signal is sent back to the source (by the end-to-end flow control) to throttle its sending rate. If the congestion is only a transient effect, it may be dealt with sufficiently by the intermediate nodes via the hop-by-hop flow control. Thus, the source may be reducing its sending rate unnecessarily producing low utilisations. Ideally, the hop-by-hop flow control should only react to shorter term congestion build-ups, whilst the end-to-end flow control should be designed to cope with longer term traffic build-ups. The current trend is for increasing control at the edge of the network, giving the user (or applications) ultimate adaptability. In the extreme case of the network only providing 'dumb fat pipes' and all control propagating to the edge, we hope that the strategies played by individual users will produce some form of global stability. This is the realm of mathematical game theory. In conclusion, we should be careful to design network controls that work on sufficiently different timescales to avoid possible control loop interaction.

1.7. Determining the Cost of Network Control

In order to compare different network control/routing techniques, we utilise some measure of performance. For example, blocking probability or packet delay are typically used for routing studies. Any network control implementation will have some cost associated with it. The three costs that are generally considered are communication, computation and storage. The costs associated with routing can be considered as an example. Unless we use a totally distributed routing mechanism, we will incur a communication overhead due to the need of the nodes to exchange state information. A route calculation

will take some computation time as dictated by the complexity of the calculation. Finally, we may or may not store state information as dictated by the computation of routes, in addition to routing tables necessary to route packets from the different flows passing through the node in question. Different routing algorithms will have different associated costs in these three areas. We are interested in how these costs scale as the network size is increased, since we are interested in network controls that will scale to the wide area. It is our belief that the most important cost is that of communication overhead. This is primarily due to the fact that with increasing bandwidth-delay products and size of networks, the primary bottleneck in communication networks will arise due to speed of light propagation delays (see [19], [20]). It is therefore necessary to consider those controls that require the minimum of inter-nodal transactions (i.e. locally based control - caching etc...) to minimise this overhead. Secondly, the number of communication messages between nodes will increase exponentially with the size of the network such that we can quickly swamp networks of even moderate size. Finally, operating a network over multiple administrative domains means that communication of state messages between these domains may not even be possible, forcing the consideration of locally based control techniques.

1.8. Summary of Network Control

Network control has been now described as a problem in space and time. In general, network control involves the three Ws, 'what, where and when' of network control. That is, what control should be used, where in the network it should be used and at what time it should be applied.

1.9. Why study routing?

Routing is a problem that captures the essence of the network control problem as described above. According to [21], routing is defined as follows:

The goal of routing in a communications network is to direct user traffic from source to destination in accordance with the traffic's service requirements and the network's service restrictions.

Routing then requires that we choose the appropriate route (what) in the appropriate place (where) at the appropriate time (when) in the network. Central to the problem of routing is how we distribute sufficient information throughout the network such that routing decisions can be made. Finally, the user traffic must be forwarded along the chosen routes. Although the controls in an integrated services network will interact with one another to some degree, we believe that routing encompasses a broad range of network control issues (i.e. the three Ws) such that insights gained by studying routing may be sufficiently general to apply to other network control mechanisms. Additionally, we believe that the problem of routing in integrated service networks may involve different assumptions regarding traffic(s) and topology to that studied previously in circuit-switched and best-effort environments, and that routing in integrated service

networks should therefore be studied in its own right.

1.10. Summary

This chapter has examined the fundamental architectural components of integrated service networks and investigated the problems that arise in the context of controlling these networks. Routing has been posed as a problem warranting further investigation since it encompasses many of the control problems discussed such as distributed versus centralised control, routing on multiple timescales, user versus network control, granularity of control and the cost of network control such as communication, computation and storage overheads.

The uncertainty regarding the traffic to be carried on integrated service networks makes it difficult to design a fixed 'optimal' control strategy. This motivates the use of 'adaptive control' where the control strategy is dependent on the demands on the network and possibly, on the dynamic network state. For general large scale networks however, mathematical models necessary for adaptive control are usually intractable. This in turn motivates the use of adaptive control strategies that may operate without a model of the environment. The field of Artificial Intelligence (AI) contains a number of techniques that can effectively control systems without the need for an explicit model of the system under control. Therefore, in the next chapter, we provide a comprehensive review of a range of AI techniques, seeing how they have been applied to network control problems.

1.11. Outline of the Thesis

The main body of the thesis is contained within five chapters. This chapter has provided an overview of the control mechanisms likely to be required in a future integrated-services network. Chapter 2 then provides a broad review of how Artificial Intelligence (AI) techniques have been applied to these control mechanisms in the literature. Chapter 3 presents the first effort to use 'Stochastic Learning Automata' (SLA) for the 'Quality-of-Service' (QoS) routing problem in future networks, whereby routes must be found within the network subject to (multiple) QoS constraints requested by the source (or receiver). In Chapter 4, automata are considered for the routing of best-effort or non-real-time (NRT) traffic. Additionally, a novel environment containing NRT and RT traffics simultaneously is considered, to investigate suitable combinations of routing algorithms for each traffic class and gauge whether one routing algorithm can be used to route both traffics satisfactorily. Chapter 5 presents the first attempt to apply learning automata to the multicast routing problem, where automata are used to set up connections in a dynamic multicast environment in order to minimise the delay and cost of the resulting trees. In Chapter 6, automata are examined for the construction of dynamic quality-of-service (QoS) multicast trees where there may be guarantees on the throughput and delay to each receiver. This represents one of the first endeavors to analyse multicast routing algorithms in a dynamic QoS bounded environment. In Chapter 7, a summary of the thesis is presented and areas for further work are identified.

Chapter 2

A Critical Review of Artificial Intelligence (AI) for Network Control

2.1. Introduction

In the previous chapter, we saw how the increasing complexity of network control functions and increasing uncertainty of the demands placed on these networks led naturally to the consideration of techniques that may control such networks without detailed models of the networks or traffic demands. The field of Artificial Intelligence (AI) has shown how algorithms based on some notion of 'intelligence' can effectively control such systems. In this chapter, we provide a broad overview of how AI has been applied to network control problems in the literature. We primarily focus on 'computational intelligence' techniques where the algorithms involved can be implemented on a computer. We are interested in those controls which operate on timescales that are too fast for human operators.

2.2. Background

AI is a relatively young field and numerous researchers across many different disciplines are performing AI related work. Many researchers will give a different definition of AI and this has caused some to question the validity of AI. At the heart of the debate lies the difficulty in defining 'intelligence' and whether the definition used is a strictly proper description of human intelligence. As far as those working in applied AI are concerned however, the real question is whether AI techniques can improve the performance of computers. As applied to the field of telecommunications and specifically, network management and control, the question should be whether AI techniques can help to contain the rapid growth in complexity of these control functions. The relative advantages and disadvantages of these AI techniques as applied to network management and control should therefore be investigated.

AI techniques can be divided into symbolic and non-symbolic approaches for simulating reasoning. The main application area of the first category has been 'expert systems'. These are defined in [22] as, 'computer systems that use knowledge and inference or reasoning procedures to solve problems that are normally handled by experts'. Expert systems were envisaged to make significant breakthroughs in the areas of network management and control during the early to mid-eighties. Typical applications are given in [23] and [24]. Many problems were encountered with expert systems however. In particular, expert

systems proved to be 'brittle' in that for a limited number of rules, a rule may not be defined for the particular input condition occurring. This can be solved by adding more rules although this can result in a proliferation of rules, introducing a 'knowledge acquisition problem'. The problems associated with traditional expert systems has led to new methods being proposed. For example, the integration of fuzzy logic and neural networks within expert systems is currently being considered (see [25]). Fuzzy logic can help overcome the brittleness problem by interpolating between rules, therefore demanding fewer rules for reasonable performance. Neural networks can also be used as generalising functions, where no hard-and-fast rules are applicable. This 'synergy' of neural/fuzzy and expert systems can possibly tackle problems that neither can solve in isolation.

Non-symbolic or computational AI can be broadly defined as those techniques which may be implemented as an algorithm on a computer. In this chapter, we cover Artificial Neural Networks (ANNs), Fuzzy Logic, Intelligent Agents and Stochastic Learning Automata (SLA). We can distinguish these control mechanisms from expert systems in that they operate at reasonably fast timescales such as the connection level, whereas expert systems are more akin to the network design level.

2.3. Artificial Neural Networks (ANNs)

Neural networks are fast becoming one of the most fertile research areas for AI in communication networks. This is because neural networks have been used to solve complex problems from the 'bottom – up' that are not easily addressed with conventional digital computers, such as pattern recognition, classification and optimisation problems. A neural network consists of a large number of basic computational elements connected in a certain topology. These elements are 'neurons' and the interconnections between the neurons ('synapses') are represented by a set of weights. Given a set of input/output data, the neural network can be trained to reproduce the mapping between inputs and outputs using some error correction learning process to adjust the weights. A comprehensive introduction to neural networks is provided in [26]. The advantages typically cited for neural networks for control purposes [27], are: (1) Adaptive Learning; (2) High Computation Rates; (3) Generalisation from Learning; (4) Fault Tolerance. Neural networks can therefore be used 'on-line' to learn the characteristics of the underlying process, generalising across state space which may not have been explicitly learnt. The high computation rates and fault tolerance properties result from the neural network's massively parallel and distributed computations. Despite these benefits, there are a number of pitfalls to avoid when applying neural networks and these include: (1) Overgeneralisation; (2) Extrapolation; (3) Selecting the right parameters; (4) Having enough data. If the neural network model is too detailed (i.e. too many nodes in the hidden layer), the neural network will learn the noise in the data set. If the network is only trained on a specialised part of the input space, it cannot be expected to give reliable outputs for inputs very different to the training set. For example, if a neural network is trained on steady state data, we cannot expect it to accurately model the process during a transient period. Input parameters to the network should ideally be orthogonal, complete and pre-processed in order to minimise duplication,

capture all significant influences and prevent reinventing the wheel respectively, and as much data as possible should be obtained so the network is able to learn the mapping required to solve the problem.

The application of ANNs to the control of communication networks can be split into two broad areas. The first area draws upon the adaptive capability of ANNs and concerns their application to some form of traffic control (CAC, policing, congestion control) where the neural network is responsible for regulating the flow of traffic into the network such that the QoS constraints of existing flows are met. The second area draws upon the optimisation capabilities of ANNs to minimise some cost function in the switching and routing problems. We review both of these application areas in turn.

2.3.1. Neural Networks for Traffic Control

For future networks, we have seen from Chapter 1 that there is likely to exist a service which gives hard deterministic bounds to incoming traffic (e.g. peak rate bandwidth allocation) and a service which improves utilisation through less severe bounds (e.g. statistical bounds). For ATM this service takes the form of 'Variable Bit Rate' (VBR) [28] and for the IETF integrated-services work, this takes the form of the 'controlled-load' [13] service, where connection admission control is performed to maintain a service akin to a 'lightly loaded internet'. Designing a fixed rule for admitting new connections to meet a certain QoS may be difficult given the uncertainty regarding the nature of future traffic. Measurement based CAC has therefore been proposed to provide a more flexible solution and measures the current average usage of the network to decide whether new flows can be admitted. For truly successful measurement-based CAC, it is thought that some form of prediction of the time-varying nature of the traffic is likely to be required. To achieve realistic predictions, high-order moments of the traffic may need to be obtained which could prove infeasible for real-time calculations. In addition, predictive based control may be more suitable for control in the future as bandwidth-delay products become large and time to react is dictated by the propagation delay. These problems have led to the consideration of ANNs for adaptive traffic control where the adaptivity, high speed properties of neural networks make for practical implementation. A modular approach to implementation has been proposed in [29] to separate the functionality of the different levels and time-scales of traffic control. Figure 2.1 shows the basic set-up. The proposed traffic control model is a hierarchical control model consisting of cell, call and network control levels with control cycle periods of sub-millisecond, sub-second and hours to weeks respectively [30]. Hiramatsu [30] has proposed the application of neural networks to integrate control at all levels in this model although no results are presented for such a scheme. The initial work of Hiramatsu [31] has looked at a simple multiplexer where a neural network creates a decision function by learning the behaviour of the operating multiplexer. A three-layer fully connected multilayer perceptron (MLP) is trained to learn the relationship between the multiplexer status (cell arrival rate is used) and the observed QoS (cell loss rate).

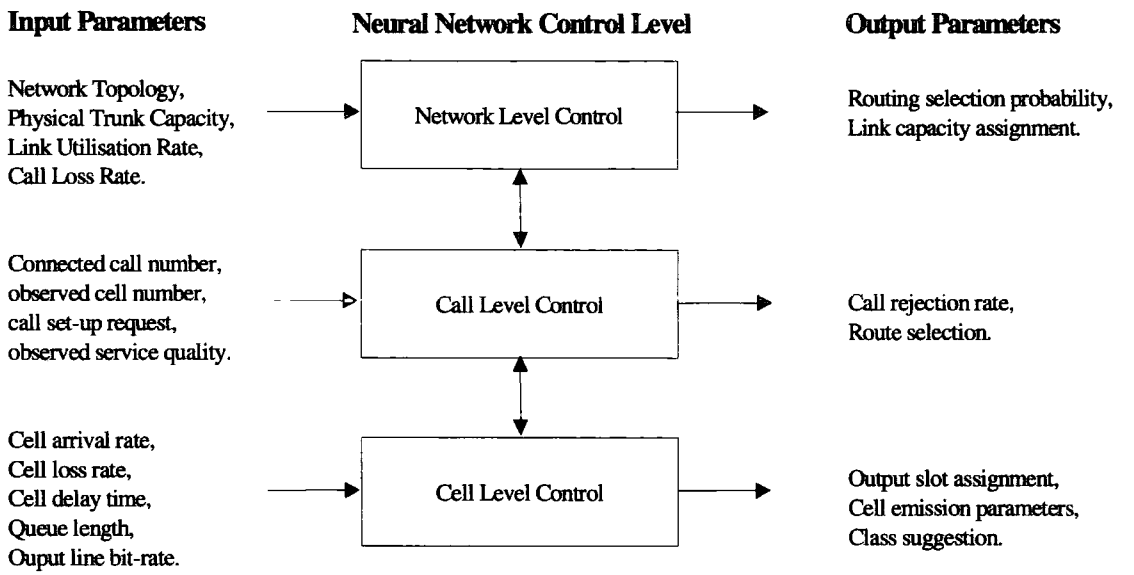


Figure 2.1 – Multi-layer Neural Network Control (adapted from [30]).

The inputs to a N input multilayer feedforward network are the number of cell arrivals in the last N time slots. A call is accepted if the output of the neural network (predicted cell loss) is below a certain threshold value. Applications to single and multi-bit-rate traffic are presented. Problems can arise when training the neural networks for loss since high loss events should be rare if the controller is operating correctly and the neural network will therefore only learn that part of the input space corresponding to low loss events. To combat this, Hiramatsu has used what is known as a 'leaky pattern table', where there are two tables, one for low and high loss rates respectively. An exemplar for training is randomly selected and an old observation is randomly chosen to be replaced by the current observation at each backpropagation step. In another paper, [32], Hiramatsu uses 'virtual buffers' with much greater cell loss probabilities to improve the training of the network. Results show that for the traffic model considered, a neural network can learn the decision boundary for admitting calls to maintain the cell loss rate below some threshold [31]. The neural network can take a relatively long time to converge, although it is envisaged that the values of the weights would be optimised through simulation 'off-line'. Essentially, neural networks applied in this way represent another algorithm for measurement based admission control, for which a plethora of algorithms have been proposed (see [33] and references therein). There is a need to compare these algorithms, possibly using real data traces. Hiramatsu [34] has extended the application of neural networks to consider the integration of call admission control and link capacity control to attempt to integrate call and network level control functions. For Hiramatsu's simulated network, each node in a simple four node model contains a neural network for call admission control. These four nodes have access to a common network control centre which contains a neural network to estimate the call loss rate of all links and attempts to minimise the maximum call loss rate throughout the whole network by altering the logical link capacities. It is shown that the link capacity allocation mechanism can adapt to varying traffic characteristics and the connection admission control. It is hoped

that the work will be extended to larger networks with multiple bit-rate classes although it is not made clear how scalability to very large networks would be achieved using the effectively centralised control proposed. Although a neural network could be constructed in hardware to give fast estimation, the on-line updating of the network necessary for 'on-line' learning could prove computationally expensive.

In an alternative CAC approach proposed in [35], for multiple traffic classes (video and file transfer are considered), the neural network learns a mapping between the number of sessions in each class and the resulting QoS (delay is used). The neural network learns an effective decision boundary for the traffic models given although simulations again concern a single node and no attempt is made to compare the approach with other techniques or utilise real traffic traces.

When setting up a connection from a source to destination, a local admission control decision is usually made at each node along the path on whether to accept or reject the session. This task has been referred to as 'Link Admission Control (LAC)'. LAC establishes whether the link should accept or reject the set-up request. If a number of links at a node pass the LAC test, the problem becomes one of 'link allocation' where one of the links must be chosen, the ideal policy typically one which maximises the total number of connections/calls accepted. One recent approach to this problem has looked at the combined use of reinforcement and supervised learning [36]. Reinforcement learning consists of some 'agent' interacting with the (unknown) environment, the agent being rewarded for actions leading to the desired effect on the environmental state (i.e. maximal reward). Supervised learning uses input/output data for the environment to 'teach' the agent the correct strategy. In [36], two neural network based schemes are presented for the link allocation problem. The first uses a technique called, 'Backpropagation with Hypothetical Targets' (BPht), where a single neural network is trained on a bipolar reward, indicating whether the link allocation was a success or failure. For each action, weight changes for the neural network are computed by supervised training on two hypothetical targets, one assuming a positive result and one assuming a negative one. The weight changes are accumulated and discounted over time and the sign of the reward indicates which one to apply when updating the weights. Results by simulation for this method [37], show that the technique has comparable performance to conventional techniques (e.g. best fit, first fit etc..), while being capable of adapting to changing traffic distributions without explicit knowledge of their distributions. The second method discussed in [36] is a temporal-difference based adaptive link allocation scheme. The link allocation task is decomposed into a set of link admission control sub-tasks. These sub-tasks are formulated as semi-Markov Decision Problems (SMDPs). The LAC policies are directly (i.e. without a model) adapted by reinforcement learning using the temporal-difference learning scheme where the reward is the aggregate cell transmission rate. Maximising the long-term reward is therefore to maximise the utilisation of the link. Results for the scheme show that it outperforms traditional static methods and has comparable performance to an indirect (model-based) adaptive method, although only simple Poisson traffic models are considered.

The ability of neural networks to learn arbitrary functions has been exploited to produce high speed

calculations of analytical functions with high computational complexity. For example, Fan has shown how neural networks can produce good estimates of cell loss calculations (i.e. QoS) for a MMPP (Markov Modulated Poisson Process) input to a multiplexer [38], and for effective bandwidth estimation [39], respectively. In [40], for the LAC problem discussed previously, an analytical bound on the cell loss probability controls the admission for low loads whereas a neural network (using estimation) controls the admission for high loads. Such an approach is shown to increase the resource allocation over solely using a neural network.

Neural networks have also been proposed as a means of access congestion control (see [41], [42]) where the communications network is modelled as a dynamic plant where the overall input (total arrival rate) to the network is regulated by the neural controller to meet some performance bound (e.g. delay). The models proposed are rather abstract and it is unlikely that an entire network can be modelled by a simple difference equation such as those proposed. In addition, there is a large body of literature concerning congestion control protocols in networks (see [43] for overview) and there is a need to compare the neural congestion control ideas with more traditional techniques (exponential back-off etc..), possibly utilising real-traffic traces.

Traffic prediction is a commonly claimed ability of neural networks (see [44] for overview). For example, [45] applies a FIR (finite impulse response) neural network to one-step prediction of video and voice traffic. The prediction capabilities are then used in a preventative rate-based congestion control scheme [46], which effectively throttles the source rate and it is shown that the neural network based control scheme outperforms a simple queue threshold mechanism in terms of overall cell loss rates. Although traffic prediction has been successful for many artificially generated traffic models, Hall [44] questions the use of neural networks at all, showing that a traditional linear regression technique works equally well for prediction of real traffic traces. This is an important result since effective traffic prediction under-pins the application of neural networks in areas such as congestion control, traffic shaping and dynamic bandwidth allocation.

Finally, neural networks have been applied to traffic policing. Connections are conventionally policed by monitoring the peak or average cell/packet rate. To react fast enough, the average calculation must be windowed over a small interval which may produce erroneous policing decisions. A neural network approach has therefore been proposed [47], using two backpropagation neural networks which implicitly learn the pdf of the traffic count process through many learning trials. One neural network is trained to learn the pdf of 'ideal non-violating' traffic, whereas the second neural network learns the 'actual' characteristics of 'actual' offered traffic. The error between the outputs of the two neural networks is fed through a cost function to a third neural network whose weights are tuned using reinforcement learning, so that the controller minimises the violations of the traffic source contracted characteristics determined at connection set-up. The technique can provide excellent policing decisions and the reaction time of the system is small compared to the window averaging mechanisms.

2.3.2. Neural Networks for Switching and Routing

An ANN approach for routing in a crossbar switch is introduced in [48]. An N*N crossbar switch has N inputs and N outputs and the switch can establish paths between inputs and outputs by control of the (N*N) crosspoints. In each row (or column) of the switch, only one crosspoint can be connected. Given a traffic demand matrix T, the objective is to maximise the number of connected crosspoints. A common ANN approach, [49], is to let a neuron correspond to each crosspoint. If a given neuron is ON, the corresponding crosspoint is closed and visa versa. An energy equation can be generated for this problem subject to the constraints above and it can be verified that $dE/dt \leq 0$ such that the energy, E, in the ANN converges to a stable state when $dE/dt = 0$. The state of the neurons at this point represent an optimal or near-optimal routing matrix. Although the Neural Network may produce only a near-optimal solution, the speed with which it arrives at a solution makes for practical hardware implementation, unlike traditional exhaustive search techniques.

For general routing in a network, Hopfield type ANNs have been proposed to solve routing problems with a similar specification to that of the Travelling Salesman Problem (TSP) [50]. Here, the Hopfield network is used to minimise a loss based cost function when the global topology and traffic matrix is known. For future networks, it is unlikely that we will be able to produce an expected traffic matrix such as that utilised. The cost function can be altered so that a Hopfield network learns the minimum cost path where cost can be defined as delay, path length etc... Essentially, a Hopfield type approach to routing represents a totally centralised technique and it is not clear how the technique presented in [50] could incorporate the decentralisation necessary for scaling to very large networks. Presumably, nodes would need to communicate so that a given node can build up a picture of the network topology and the link costs, this information being translated into the appropriate Hopfield network. The number of iterations for the Hopfield network to reach a steady state solution as the communication network size is increased needs to be investigated and compared with more traditional shortest path algorithms with known computational complexities (e.g. Dijkastra's algorithm). In [51], a feedforward neural network is used at each node in the network and uses Hebbian learning to update the weights, where the feedback to the neural network is the cost of the path that messages were routed along. The scheme is truly distributed and the number of messages sent to find the minimum cost routes has a worst case scaling of $O(N^3)$ for an N node network. The scheme is shown to find the minimum cost paths fairly reliably although it is similar in many respects to the application of traditional reinforcement learning to the routing problem (see section 2.6.3.).

In [52], a set of distributed neural networks is used to minimise the total cost where cost is defined as the weighted traffic delay. The neural networks learn an appropriate mapping between the state of the queues and the next node to send traffic to. It is shown through simulation that neural networks using purely local information produce a cost only slightly greater than that produced when global information (of queue states) is assumed.

To minimise the cost of a multicast tree, a recurrent Random Neural Network (RNN) has been proposed in [53]. The neural network starts with the solution of typical Steiner tree heuristics and perturbs them to find potential Steiner vertices that are not already in those solutions. The excitatory weights in the neural network are inversely proportional to the edge costs such that neurons have high excitation for lower cost connections. The RNN approach further reduces the cost from the heuristics, although the approach is a centralised method for determining minimum cost trees and a distributed scheme will be required for multicasting in practical communication networks (see Chapters 5 and 6). Since the minimum cost multicasting problem is similar to the Travelling Salesman Problem (TSP), those neural network structures proposed previously for solving the TSP could be easily modified to solve the minimum cost multicasting problem.

One final application of Hopfield optimisation networks has examined their potential for scheduling of packets which arrive at the input queues to a crossbar switch. Here, a neural network typically selects packets for transmission through the switch based on some window of packets in the buffer. The idea is to prevent 'Head of Line' (HOL) blocking whereby, if the first packet cannot pass through the switch, it blocks packets behind it which may potentially pass through the switch. Simulation studies (see [49]), show that a neural controller can produce throughputs within 1 or 2 percent of those produced by exhaustive search, even for large switches. In [35] however, a simple heuristic scheme is shown to have competitive performance, questioning the use of neural schemes at all.

2.4. Fuzzy Logic

Since the conception of Fuzzy Logic by Zadeh [54], in 1965, the range of application of fuzzy logic has increased considerably. The basic idea behind fuzzy logic is that it provides a framework for dealing with imprecision. The application of fuzzy logic to control is known as, 'fuzzy control'. The basic operation of a fuzzy controller is shown in functional form in Figure 2.2.

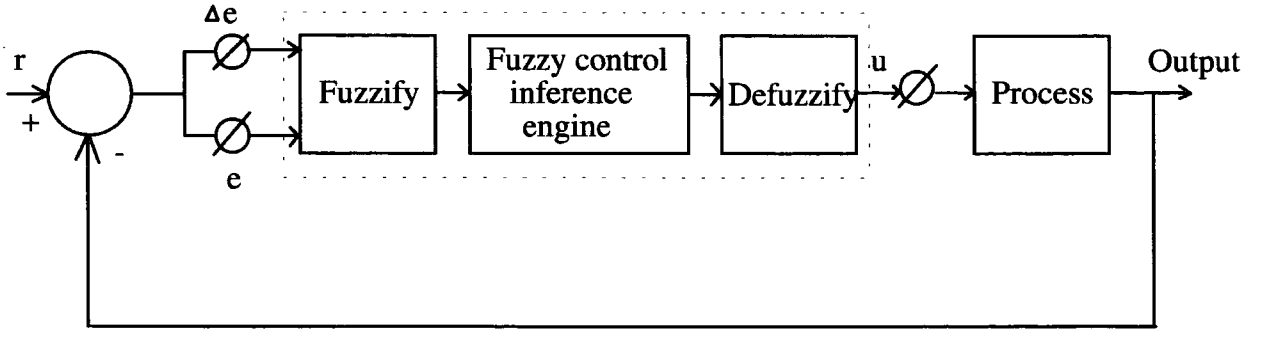


Figure 2.2 – Typical Fuzzy Control

The continuous outputs from the process are 'fuzzified' or mapped onto 'fuzzy sets', which are continuous function of these outputs. The knowledge of an operator is contained within a 'fuzzy rule base' and contains so called 'rules of thumb', so that the operator's knowledge can be defined in a

heuristic and imprecise way. Hence, functions are performed on the fuzzy sets as defined within the rule base and the resulting fuzzy sets are 'defuzzified' to produce a 'crisp' continuous input to the process under control. Advantages of fuzzy control are the ability to encapsulate linguistic information, the lack of a need for a plant model, controller robustness and a good performance history in the process control industry. Development of fuzzy control has been largely application driven and there are a number of drawbacks with fuzzy control including difficulty of tuning and lack of a design procedure, lack of fuzzy control theory and no 'on-line' learning capability for classical fuzzy systems. It has recently been shown that there is a one-to-one mapping between certain fuzzy systems and neural networks [55]. This means that a system could be designed based on fuzzy rules but converted to a neural network which has on-line learning capabilities.

The application of fuzzy logic to network control has not received the same degree of attention as neural networks. However, the application areas such as admission control, policing, congestion (rate) control and buffer management are similar to those studied with neural networks.

2.4.1. Fuzzy Logic for Traffic Control

Connection admission control is considered in [56], where a fuzzy controller estimates the upper bound of the cell loss ratio (CLR) in order to accept or reject the current connection request. The fuzzy rules take the number of connections in each traffic class to estimate the CLR. Also, an on-line tuning algorithm based on back propagation is used to tune the fuzzy set widths based on the observed data. It is shown that the fuzzy controller effectively learns the upper bound on the CLR and extrapolates the fuzzy rules to areas of the state space for which no observations are available. Only one traffic class is considered in simulations and 7 fuzzy rules are used. One problem with fuzzy control here is that the number of rules required increases exponentially with the number of input traffic classes (i.e. curse of dimensionality). Also, enough expertise about the system must be available to formulate the rules. In [57], a fuzzy CAC controller is designed based on Guerin's effective/equivalent bandwidth approximation. The algorithm is shown to give higher utilisations since it uses a measure of the dynamic network state (cell loss) in addition to the connection traffic parameters.

A fuzzy policer is designed in [58] based on policing the average connection rate. The inputs to the controller are the number of cells since the call establishment, the number of arrivals over the last T secs and the number of cells, N, that can be admitted in the course of a second window period. The output is the necessary change to N. 18 rules are used by the fuzzy controller which is shown to be superior to conventional window based techniques via simulation using real traffic traces. Fuzzy logic has also been proposed for the emulation of the 'Leaky Bucket' policing mechanism that involves choosing a buffer (bucket) of finite size (or finite number of tokens) to contain bursts of cells, which then drains at a constant rate. One particular variant of the 'leaky bucket' mechanism is the 'virtual leaky bucket' (VLB), which permits user excursions above the negotiated rate but marks the violating cells so that they may be readily discarded later. A fuzzy implementation of the VLB is presented by Ndousse [59] and is thought

to be beneficial, since the operation of such a mechanism can typically be defined in a vague or 'fuzzy' manner since it is a complex control problem. The idea is to articulate expert knowledge about the mechanism into a set of fuzzy rules, in an attempt to improve the performance of the controller. Ndousse [59], uses a two input, single output fuzzy controller, where the inputs are taken from the token and QoS buffers and the output is the number of cells which may be tagged (for eligible discard) in the next time interval. The fuzzy rules are stored in a 'fuzzy associative memory' (FAM) to permit parallel execution. Traditional and fuzzy leaky bucket methods are simulated using a Markov Modulated Batch Poisson process (MMBP). Results from [59] for the fuzzy controller show that the fuzzy leaky bucket mechanism gives higher throughput performance than traditional leaky bucket methods and it is stated that performance can be further improved after additional tuning. The issue of tuning is not addressed however, this being a critical factor when considering any fuzzy control application. Further work includes examining the viability of silicon implementation and the possible use of adaptive fuzzy algorithms for improved (learning) performance.

In [60], Bonde and Ghosh present the idea of cell-blocking for cell-switched networks, where a number of incoming cells are blocked or refused entry to the buffer so that a trade-off between the number of cells carried through the network, propagation delays of the cells and the number of discarded cells may be obtained. In the fixed threshold case, the buffer will only accept cells from an input burst if the occupancy of the buffer is below a certain threshold. Bonde and Ghosh propose a fuzzy thresholding function which deliberately blocks a fraction of incoming cells from other switches, so that there is an increasing probability of blocking as the occupancy of the buffer increases. A simple cell-blocking scheme has been simulated based on Poisson arrivals for a single server queue and exponentially distributed departures and fuzzy and binary cell-blocking approaches are contrasted. The fuzzy method is shown to achieve higher throughput, lower discard rates and lower cell blocking rates. There are therefore plans to extend the work to a 50-60 switch network.

2.4.2. Fuzzy Routing

A recent paper by Chemouil, Khalfet and Lebourges [61], has examined the viability of a fuzzy control approach for traffic routing in circuit switched networks, which is applied to a model of the French exchange. Here, circuit groups are chosen on the basis of availability using rules like, 'IF number of idle circuits is LARGE AND number of in-service circuits is SMALL THEN availability is MEDIUM' for each circuit group and the availability indicator is based on the classical residual capacity (least loaded) approach. The availability for each (two leg) route is determined and the fuzzy quality is calculated for every route. The paths are ordered in decreasing order of route quality and the best path is chosen to route the calls for the next time period. The proposed fuzzy scheme is compared to fixed and idle capacity routing schemes, by analysing call loss rates for varying traffic loading and sampling times. Although the fuzzy approach outperforms the fixed and idle capacity schemes, in that lower call loss rates are achieved, the gain over the idle capacity approach is rather small. To improve the performance of the

fuzzy control technique, the authors recognise that the probability that a circuit group becomes saturated is not simply a decreasing function of the residual capacity, but also dependent on the offered traffic. A new availability indicator is therefore defined based upon both the residual capacity and the carried traffic and the performance of the fuzzy controller is seen to increase considerably, keeping network loss rates low even for infrequent sampling. When this new indicator is used in a classical (non-fuzzy) way as an indication of residual capacity, extremely poor results are obtained. Thus, obtaining the capacity indicator is a critical factor and not straightforward. Also, a description of the tuning process is omitted which can be the most difficult aspect when implementing fuzzy control. Further work is intended to develop the fuzzy routing scheme for different network structures and evaluate the performance.

In [62], a two layer hierarchical fuzzy system is considered for adaptive routing in a fully connected 8 node network. There is a network control centre (NWF) which is connected to the local controllers at each node (NDF). The NWF provides reference routes and the NDFs calculate alternate routes based on the number of free trunks, the recommendations of the NWF and the revenue of a call. It is shown that for uniform and localised loadings, the fuzzy scheme maximises revenues over more traditional state dependent routing schemes. It is not clear how this scheme will scale to arbitrary large topologies since it requires significant information about the dynamic network state. Also, a large number of rules (requiring sufficient knowledge of the process) are required, defeating the point of the extrapolation capabilities of fuzzy control to some extent.

2.5. Intelligent Agents

The term 'intelligent agent' has recently become very popular, being used to describe anything from World Wide Web (WWW) browsers to mobile robots. Here, we briefly review some of the different types of agent currently being investigated. We conclude that so called 'reactive agents' hold the most potential for communication networks, enabling a distributed control paradigm that allows fast acting localised control based upon the construction of a rule-set to produce a set of 'task achieving behaviours'.

2.5.1. Taxonomy of Intelligent Agents

A recent paper by Nwana [63], provides a comprehensive overview of the agent research area. He identifies seven types of agents: 1) Collaborative agents; 2) Interface agents; 3) Mobile agents; 4)Information/Internet agents; 5) Reactive agents; 6) Hybrid agents; 7) Smart agents. Nwana distinguishes those applications which combine agents from two or more categories, these being identified as *heterogeneous agent systems*. From a network control perspective, classes 3, 5 and 6 are of particular interest. Consequently, we give an overview of these three types of agent.

2.5.2. Mobile Agents

Mobile Agents are entities containing code and data which are executed at a remote site. They are capable of roaming networks to perform some function on behalf of the user. As pointed out in [63],

'functions are shipped to the data', such that an agent can gain access to data and resources in the most suitable place in the network. A classic example of the possible benefits of the mobile agent approach is given by Nwana in [63]. Suppose we wish to transfer a graphical image from a remote site. This image is one of say two hundred, that we must search through to find the particular one we require. With a conventional approach, we would transfer each of these images from the remote site to our local host and see if we had the correct one. With a mobile agent approach, we would dispatch an agent containing a search routine to the remote site. The agent would execute this search routine at the remote site identifying the image we require. The correct image would then be dispatched to our local host. We see from this example that we gain no new functionality (i.e. we can eventually receive the image with or without mobile agents) but that we receive the image at a much lower cost, both in terms of communication cost (i.e. bandwidth usage) and lower use of our local (and maybe highly resource limited) host.

In the network control area, Appleby and Steward [64], have investigated the use of mobile agents for adaptive routing purposes. Their scheme is based on three basic principles, which they propose will achieve robustness: 1) there should be no direct inter-agent communication; 2) the agents should be present in reasonably large numbers; 3) the agents should be able to dynamically alter their task allocations and number. The system should therefore be robust to failure of one or more agents since agents can operate independently of other agents and we also have safety in numbers. The third rule ensures that we have 'fit for purpose control' in that a set of agents can adapt (population and task division) to the problems at hand. It should be noted however, that the agents can only adapt as specified in the rules for the behaviours of the agents. For the system to autonomously change these rules, we require some form of learning or evolution (i.e. the rules for changing the rules etc...). Appleby and Steward draw upon these three principles using Brooks's subsumption architecture [65] to guide the design of their agent system. There are two types of agent in their system, 'load agents' and 'parent agents' which provide two different layers of control in the network. The load agents utilise a clever modification of Djkastra's shortest path algorithm to find new routes. The mobile parent agent is a level above the load agent and is responsible for managing the population level and task allocation of the load agents and ultimately, to balance the network load. So called 'parent monitors' are static processes fixed at the nodes of the network and are responsible for managing the population of parent agents in order to replace crashed agents and build up the level of parent agents when the network is initialised. When the mobile agent approach is applied to a typical 30 node network, the agent scheme is found to improve load distributions compared to a shortest path routing approach under steady state conditions. Results showing the dynamic behaviour of the agent scheme are not given however, which will be important, since there will be some time lag for the mobile agent scheme to react to changes in traffic loading.

Of particular interest with the mobile agent scheme proposed is that it provides a neat compromise between distributed and centralised control methods. Distributed control is beneficial since it is generally more robust and faster reacting then centralised control, although a centralised controller has global state

information available to it and can therefore generate control actions which are closer to the optimum, albeit with a time lag from sequentially interrogating the neighbouring nodes. A combination of distributed and centralised control is likely to produce the best results achieving a compromise between speed of response and optimal control actions. The mobile agents proposed by Appleby and Steward achieve such a compromise by using an exponential averaging mechanism to vary the number of nodes visited by the parent agent to gather data.

Recent work has built on the original research by Appleby and Steward, investigating the use of 'ants' for the adaptive routing problem. In [66], ants are shown to achieve load balancing in a 30 node network, since as they randomly move about the network, the ants leave a 'pheromone' trail as a function of their distance to the source node and the congestion level on the path. The ants choose the strong trails with higher probability and the system converges producing a load balancing effect. The ants improve blocking probabilities over a shortest path scheme and the previous mobile agent technique. It is not clear how ants improve over existing protocols which distribute link state information in that, the ants effectively represent a communication process between the nodes and an analysis of the computational, storage and communication overheads is not given. In [67], a more extensive ant scheme is proposed. Here, each node periodically sends an 'agent' (a packet) to each destination and travels to the destination node based on probabilities stored at the intermediate nodes. The probabilities are updated using a reinforcement learning scheme using delay as the performance metric. The scheme is almost identical to previous adaptive routing schemes based on the use of stochastic learning automata (SLA) (see later in this chapter) and it is shown for an irregular 14 node network under static and dynamic traffic demands, that the scheme considerably improves average delay performance over current shortest path schemes. (i.e. OSPF (Open Shortest Path First) [68]). Again, communication, computation and storage complexities are not given. Also, the work could be extended to consider the interaction with congestion/flow control since it would be possible to 'piggy-back' the delay feedback onto the end-to-end flow control signals.

Mobile agent ideas present the possibility of using more complex, perhaps 'intelligent' packets, that the static processes at a node may utilise. These packets differ from existing ones in that they can be executed at remote nodes to achieve a number of behaviours in order to build up a partial picture of the network state and would probably work asynchronously and independently of one another for robustness purposes. Most current routing protocols use 'dumb' packets to signal a binary response or local connectivity information which is flooded to other nodes in the network. For example, rather than flood state information throughout the network blindly, an 'intelligent' packet may have sufficient intelligence to make decisions on which nodes to visit to gather the most relevant state information for the sending node. The static intelligence at a node may operate without 'intelligent' packets but we would expect an improved performance when the static processes utilise the additional state information stored in an agent. 'Intelligent' packets will probably be larger than traditional signalling packets since they would contain code as well as data to be executed at a remote platform. Really then, we are trying to optimise how

much information should be stored in a packet such that we can optimise the performance of a network control function. These ideas are similar to those starting to appear in the field of 'active networks' [69].

2.5.3. Reactive Agents

Reactive agents are a special type of agent that do not generally possess internal, symbolic models of their environment. Instead, they use a 'sense-act' approach to respond to the present state of the environment in which they operate. One of the greatest proponents of the reactive approach is Brooks [65]. Even though agents may contain only very simple rules for interacting with the environment and each other, they can display relatively complex behaviours. This is accredited to the fact that they reflect the complexity of the real world rather than any intrinsic complexity of their own. It is believed that surprisingly complex organisms (or 'super-organisms') in nature make extensive use of such reactive mechanisms. Whether reactive systems can be described as truly 'intelligent' is debatable although they may have many attractive properties when developing control strategies of large-scale systems. In particular, they are regarded as more robust and fault tolerant than a large centrally controlled resource although the rules or behaviours are not always easy to derive.

Firstly, we describe in more detail the typical characteristics of a reactive architecture. Reactive architectures are generally also referred to as 'horizontal architectures' in that, all layers of an agent have access to both perception and action components. In Brooks' 'subsumption architecture', based on the inputs to the layers (or modules), which themselves are groups of augmented finite state machines (AFSM), higher layers may inhibit or subsume lower layers. Each layer has a certain behaviour encoded within a rule base, e.g. avoid obstacles or random wander. In Figure 2.3, taken from [65], Brooks compares the subsumption architecture with a traditional controller.

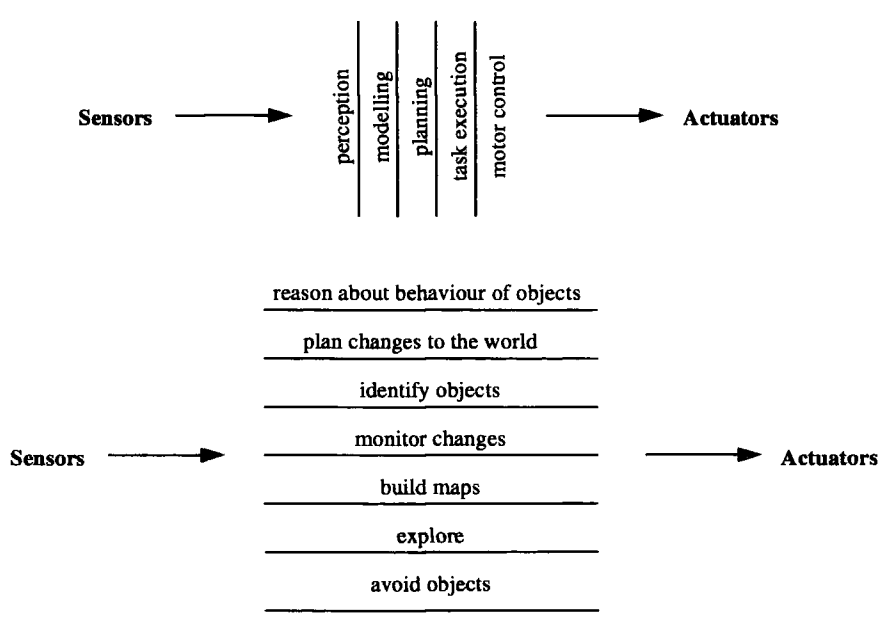


Figure 2.3 - Traditional (top) and subsumption (bottom) controller architectures.

The main advantages of reactive agents is that they can be more robust and fault tolerant than other agent-based systems, since agents can now be lost without detrimental effect to the entire system. Consequently, they are quite flexible and adaptable, despite only having fixed rules for their behaviours and interactions. The main disadvantage of reactive systems is that it is not always obvious how to design the rules for the architecture so that the intended behaviours emerge from the interaction of all the discrete parts.

In nature, reactive actions or impulses are associated with extremely fast or instantaneous responses. Similarly, for control of communications networks, we will want to react to local disturbances as fast as possible, preferably without consulting other nodes in the network since this introduces further propagation and processing delay. We therefore envision that reactive systems will form the lowest layer of control at the nodes and that it will act locally without consulting other nodes (e.g. a network resilience function). This is not to say that the reactive layer will not use more global information to influence how it will react, indeed, higher layers may gather such information and pass it to the reactive layer. The point is that the reactive layer will not rely on information from other nodes. Some would argue that reactive control will not be sufficient for future networks due to increasing bandwidth-delay products and that some form of predictive or preventative control is necessary. We argue however that due to the complexity of such systems, it is unlikely that we will be able to predict with any accuracy over long enough timescales to make prediction useful. (see section 2.2.1. on Neural Networks for traffic prediction)

2.5.4. Hybrid Reactive Agents

The idea behind hybrid agents is simple - to draw upon the strengths of different agent architectures to best match the application they are to be applied to. Most hybrid agents have focused on the combination of the deliberative and reactive paradigms. The idea is that the reactive component handle unpredictable events with the benefits of robustness, faster response times and adaptability. The deliberative part would handle longer term goal oriented tasks. Some examples of hybrid architectures are presented by Muller et al in [70], and by Ferguson in [71]. Here, we describe the architecture proposed by Ferguson in more detail since his hypothesis on agent design is close to our views regarding the future of network control.

2.5.4.1. Touring Machines

In his thesis, 'Touring Machines: An Architecture for Dynamic, Rational, Mobile Agents', Ferguson describes his agent architecture in detail [71]. It consists of three separate control layers : a *reactive* layer R, a *planning* layer P, and a *modelling* layer M. These three layers are concurrently operating, each having access to perception apparatus. Touring Machines is therefore a good example of a *horizontal* control architecture. Each of the Touring Machine's activity-producing layers is designed to cope with the world at a different level of spatio-temporal abstraction. Specifically, the reactive layer provides fast reactive capabilities for coping with immediately pressing events (which higher layers could not predict

or plan for). The planning and modelling layers are deliberative based reasoning layers where plans and models are constructed to attempt to achieve high level goals or aims. The three control layers are mediated by a *control framework* so that an appropriate layer passes its recommendations to the system output. The Touring Machine agent control architecture is shown in Figure 2.4 below.

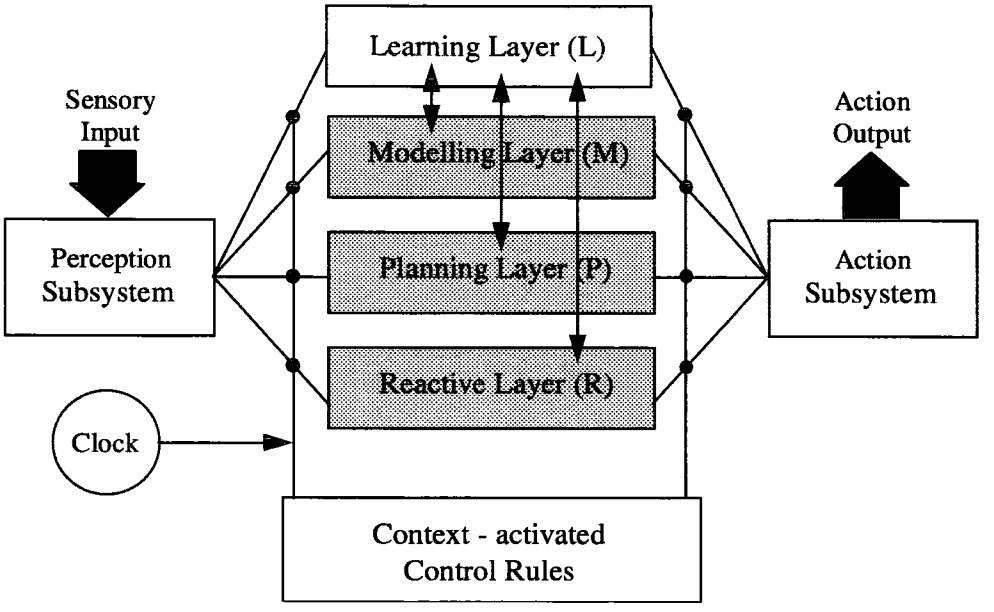


Figure 2.4 - Touring Machines Agent Control Architecture

The control framework consists of the perception and action subsystems, inter-layer message passing and a set of control rules dictating when each layer should have its recommendations passed to the action subsystem. Generally, each layer will have a range of possible outputs to choose from, which could be continuous or discrete. For the network routing problem for example, these outputs could take the form of output routing probabilities. Thus, each layer will recommend a particular set of routing probabilities. It is the job of the control rules to choose between these probability sets.

2.5.4.2. Addition of Learning to Reactive Architectures

In the summary and conclusions section of Ferguson's thesis, he suggests adding a learning layer to the architecture as shown in the top layer of Figure 2.4. This learning layer can essentially provide learning in 2 contexts. Firstly, the learning layer can form a secondary feedback loop by using some form of reinforcement learning. It then becomes possible to analyse the performance of the existing three control layers and the control framework and modify them accordingly by changing parameters for example (as opposed to *inventing* new rules - i.e. simply altering the parameters in the current rule base). Secondly, this learning layer may provide useful control actions in its own right. Thus, utilising reinforcement learning, the layer continually recommends control actions, the same as the other layers, and the control rules may choose to output the recommendations of this learning layer in preference to other layers, depending on the nature of the control rules and the feedback they receive. In the following section, we

propose that learning layers could be constructed from traditional stochastic learning automata (SLA) or Neural Networks.

2.5.5. A Proposed Agent Structure

Here, we show how an agent structure such as that discussed above might map into network control. We consider the problem of network routing as a specific example. The agent structure is shown in Figure 2.5.

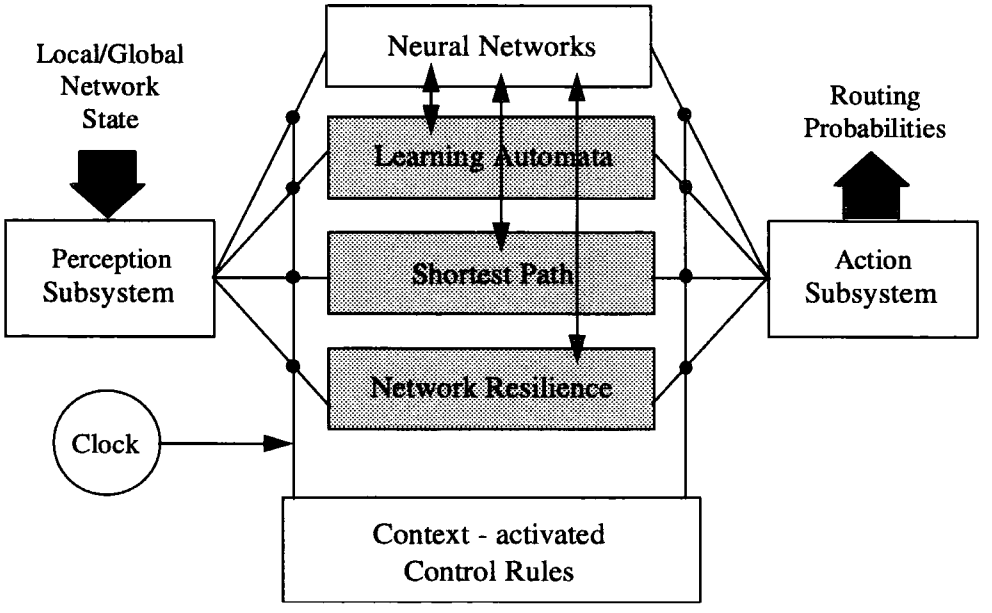


Figure 2.5 - Proposed Control Architecture

In terms of the routing problem, each of the layers in Figure 2.5 can be considered to suggest a possible set of routing probabilities to the output. At the lowest timescale, we have a network resilience function which has rules to deal with link/node failures. At the routing level, we have traditional shortest path routing in addition to learning automata based routing, which can operate concurrently to the other layers, learning a good route set. The highest layer might contain a neural network that learns a mapping between the current traffic conditions and the most appropriate layer below to pass its routing probabilities to the output (i.e. traffic classification). There could of course be additional layers added to this set-up. For example, another layer could contain route sets envisaged at the design stage based upon some measure of the expected traffic demands on the network. As seen in Figure 2.5, there is also potential for communication between the layers. For example, the learning automata and neural network layers which are learning the network state, may pass routes to the resilience layer to be used in the event of a network failure. The outputs from each layer are fed to the set of control rules which decide which layer (set of routing probabilities) should be active based on the current network state. In other words, the control rules decide which set of suggested routing probabilities should be used at any instant in time.

The problem then is to formulate a set of sufficient control rules such that we switch between layers at the appropriate time in the appropriate place in accordance with the changing traffic levels. In particular,

for the routing problem, we require that all nodes are adopting a similar routing strategy at any instant in time to provide coherent (loop free) routes. Also, what information should be fed back from the environment for the control rules to make such a decision? Specifically, should this information represent the network state local to this node or should it be more global in scope? This is important since even though the state may be changing locally to a node, it may not be a good idea to change the control strategy from a global perspective. Thus, we must weigh up both local and global (or partial global) changes in network state to determine which control strategy we should adopt. In general, the proposed architecture can be thought of as a scheme for integrating multiple control techniques. For example, different routing algorithms can be encapsulated within different layers of the architecture. The difficult part is to formulate a set of control rules dictating when one algorithm should be used in preference to another.

2.6. Stochastic Learning Automata (SLA)

A Stochastic Learning Automaton (SLA) can be defined as an 'agent' with a finite number of actions interacting with a random environment. Each action is assigned a probability and the random response from the environment resulting from choosing an action is used to update the probability that the action is selected in future using some (reinforcement) learning rule. In this way, the automaton learns asymptotically to select the optimal action. The theory of SLA is summarised in Appendix A. SLA are particularly attractive due to the sheer simplicity of their operation and the intuitive mechanism by which the probabilities are updated, that is, actions leading to favourable responses are rewarded and those leading to unfavourable ones are punished. Additionally, they are particularly well suited to the control of distributed systems such as communication networks, an automaton usually located at each node in the network. They therefore offer an alternative control paradigm to the typical centralised controller which gathers state information for the entire network and implements the controls after an inherent time delay. Furthermore, very little prior information is assumed about the environment, the automata not requiring knowledge of the expected demands to be placed on the system. The simplistic nature of the automaton also leads to some of its drawbacks. Firstly, previous studies have found automata to have a slow rate of convergence since the automata use very little prior information and operate without any direct inter-automaton communication. This problem is particularly acute when each automaton has a large number of actions and/or there are a large number of decision makers (i.e. total automata). However, as described in Appendix A, a number of recent developments of automata theory have proposed new automata models which aim to improve speed of convergence whilst maintaining the asymptotic properties of traditional models. (e.g. discretised automata, estimator based automata) In a practical application, automata are also likely to be used as a complement to existing techniques such that it becomes possible to incorporate prior information and operate automata as a background learning process. (see Chapter 3) In some situations, it can be difficult to identify a suitable index of performance to feed back to each automaton. This feedback is critical to determining the success of learning automata in a given

application.

For application to network control, automata have primarily been proposed as a means of adaptive routing in environments with very little information. However, applications to flow control and queuing systems have also been proposed and the relatively recent interest in QoS is spawning new application areas. We firstly review the use of automata for flow control and in queuing systems and then document their application to routing.

2.6.1. Flow Control

During excessive loading in packet switched networks, it can become necessary to regulate entry of traffic into the network to prevent serious congestion from occurring. Learning automata approaches to flow control are summarised in [72]. For so called 'isarithmic' schemes, each message must secure a permit before entry into the network. For asymmetric traffic, permits may become pooled at certain nodes in the network producing unfairness. Learning automata based adaptive control schemes have therefore been proposed. Two cases in [72] are discussed, derived from the work by Mason and Gu [73]. The first is a centralised adaptive flow control scheme. Here, the centralised controller attaches a permit to each message from a message source, noting the node identity to which the message is routed and giving the permit a sequence number for later identification. Each path of the permit is associated with an action and the automaton updates the action probabilities when permits pass through it. In the second case, a decentralised flow control scheme is described where N automata are associated with the N destination nodes. Each automaton has (N-1) actions, one for each of the other nodes, and each automaton operates as before. The objective is to optimise the flow of traffic through the network. This will generally depend on the number of permits used, the nature of the traffic and the routing algorithm used [72]. Simulation studies have been performed for small networks, since the optimal permit size and allocation can be determined by exhaustive search for these cases. Results from [72] for the centralised scheme with various performance feedback (loop population, loop delay and loop power) and different traffic conditions and permit sizes are plotted against network power (throughput/delay). Although close to the optimum, different feedback schemes give varying performance depending on permit size, traffic and network type. A combination of feedback measures would therefore be desirable depending on network conditions.

2.6.2. Queuing Systems

Learning automata have been applied to queuing related problems such as priority assignment, control of service activity and task scheduling in computer networks. For the priority assignment problem, the actions of the automaton corresponds to selecting a class of queue with a certain probability and the probabilities are updated based on whether the delay is below some threshold [74]. Automata have also been used to regulate entry of packets into a single server queue in order to maintain spare capacity to other traffics without explicitly using a priority queuing mechanism [75]. In general, automata can be

applied to the 'joining the right queue' problem where the automaton probabilities dictate which of several queues to join in order to minimise some performance index such as overall packet delay. The use of automata is useful here since they will learn to split the load in the right proportions without *a priori* knowledge of the system parameters (i.e. arrival and service rates). The section on routing contains an example displaying the benefits of learning automata for a simple two path routing problem which is equivalent to a 'joining the right queue' problem with two queues.

In a more recent application of learning automata to scheduling, Hall [76] uses a learning automaton to select a traffic stream to serve when a single multiplexer is fed with multiple traffic streams having different QoS requirements (delay is used). The probabilities of the automaton correspond to choosing a particular queue for service and the probabilities are updated at the expense of a best-effort queue based upon whether a particular stream is meeting its delay requirement. The scheme is shown to be able to meet the delay requirements of the respective streams whilst maximising the available capacity to the best-effort traffic.

2.6.3. Routing

The routing area has probably seen the largest application of learning automata since it is inherently suited to some form of distributed control. The use of learning algorithms is motivated by the need for lightweight (overhead) adaptive routing mechanisms which make efficient use of network resources without requiring knowledge of the network topology or underlying traffic statistics. Routing studies have generally focused on circuit-switched (see [77], [78], [79], [80], [81], [82]), virtual circuit ([83], [84], [85]) and datagram routing ([86], [87], [88], [89], [85]). Simulations on small scale networks where optimal routing schemes are tractable have verified theoretical models of learning automata behaviour (see Appendix A) and showed that learning automata based routing yields a performance close to optimum. Part of the motivation of this thesis was to look at the performance of learning automata based routing in larger networks with more realistic topologies to see the type of performance gains achievable over routing mechanisms that are used today. We also consider the effects of routing in a truly integrated-service environment in Chapters 3 and 4, considering issues such as routing of QoS based flows, aggregation, granularity of flows, delayed feedback (high bandwidth-delay products) and routing of multiple traffic types. The relevant papers are reviewed in more detail in the appropriate later chapters.

To display the benefits of a learning automata based approach to routing, we present a simple two path routing problem for which, optimal routing solutions are realisable. The set-up is shown in Figure 2.6.

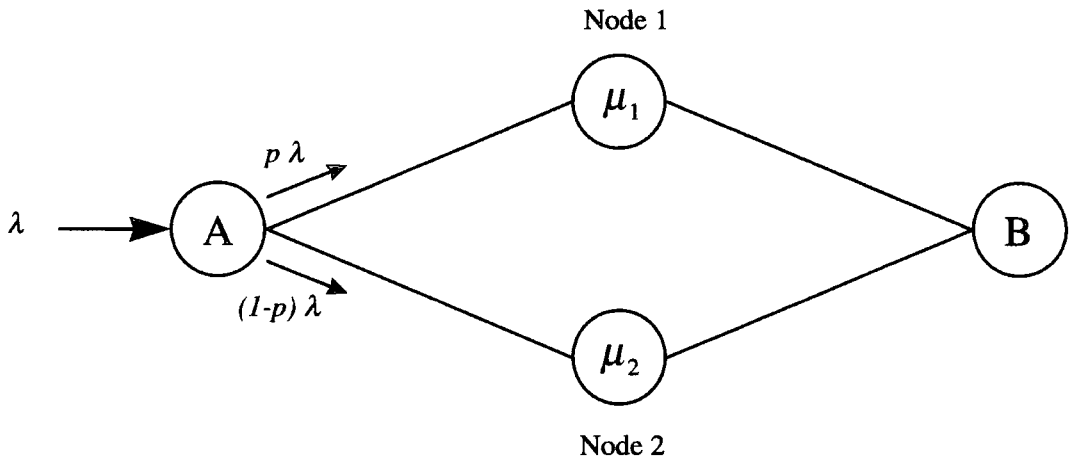


Figure 2.6 - 2-Path routing problem.

A Poisson traffic stream of rate λ enters at node A destined for node B. Node A has a choice of two paths, one via node 1 and one via node 2. Nodes 1 and 2 both maintain queues with infinite buffers and service rates of μ_1 and μ_2 respectively. Assuming that the incoming arrival rate is less than the sum of the processing rates, $\lambda < \mu_1 + \mu_2$, and the faster server is $\mu_1 \geq \mu_2$, the optimal static probabilistic split, p, to achieve minimum expected packet delay can be shown to be [90] :

Case 1: if $\lambda \leq \mu_1 - \sqrt{\mu_1 \mu_2}$, we send all traffic to the faster server so that $p = 1$.

Case 2: else, $p = \dfrac{\sqrt{\mu_1}\left[\lambda - \left(\mu_2 - \sqrt{\mu_1 \mu_2}\right)\right]}{\lambda\left(\sqrt{\mu_1} + \sqrt{\mu_2}\right)}$. (2.1.)

Thus, in order to calculate the static optimal probabilistic split, we require knowledge of the arrival rate λ, and the service rates μ_1 and μ_2. In addition, by definition, a static split will be unable to adapt to changes in these parameters. In general, optimal static policies will be inferior to those policies which use dynamic state measurements to influence their decisions. In [91], it is shown that the optimal dynamic policy for a similar example is a *threshold* policy. Here, the idea is to send packets to the faster server unless the difference between the size of the fast and slow server queues exceeds some threshold. The optimal threshold will depend on λ, μ_1 and μ_2. Thus, to execute the optimal dynamic policy, node A must have knowledge of λ, μ_1 and μ_2 and the current queue lengths or state at nodes 1 and 2. Given that the queue lengths at nodes 1 and 2 are given by $q(1)$ and $q(2)$ at any instant in time, for an arbitrary threshold a, we send an incoming packet to the faster server providing :

$$q(1) - q(2) \leq a \tag{2.2.}$$

It should be noted that the optimal dynamic policy for scenarios any more complicated than that presented here is usually intractable [91]. We assume here that node A has instantaneous knowledge of the queue

lengths, $q(1)$ and $q(2)$. In practice, node A would have to communicate with nodes 1 and 2 to acquire this knowledge using some form of control packet. This would mean that the state information is delayed and the control packets would add to the delay of the data packets themselves.

The final approach that we discuss uses a learning algorithm to split the traffic between the two servers. Specifically, a *'Stochastic Learning Automaton' (SLA)* is used to probabilistically split the traffic. The automaton uses knowledge of the packet delays between nodes A and B to update its probability p. The automaton does not require knowledge of the incoming traffic rate λ, the service rates μ_1 and μ_2 or the dynamic state of the queues, $q(1)$ and $q(2)$. Stochastic Learning Automata are discussed in more detail in later chapters (also Appendix A), here we simply wish to demonstrate the attractive properties of adaptive/learning control. In practice, node A will learn of the packet delays through node B sending control packets back to the source. These control packets will in turn add to the delay. Here, we assume an idealised case where the automaton at node A receives instantaneous feedback regarding the end-to-end delay of the data packets sent.

The three control approaches described above have been simulated for the 2-path routing problem to obtain the steady state packet delays. In Figure 2.7, we show plots of steady state average delay against packet arrival rate, λ, for the optimal static policy, learning automaton and a dynamic policy with thresholds, a, of 2 and 5. 90% confidence intervals are shown.

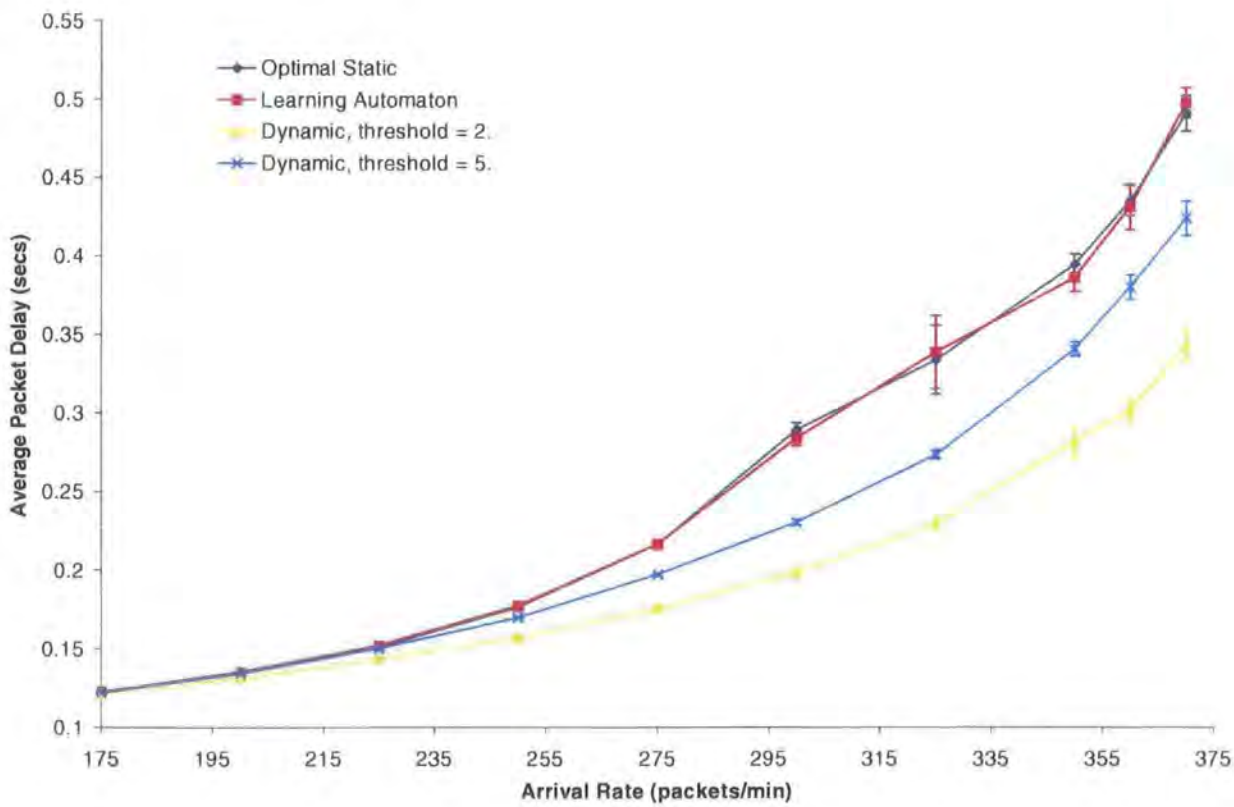


Figure 2.7 - Average packet delay, 2-path routing problem.

In the steady state, the learning automaton should never outperform the optimal static approach and the fact that the confidence intervals of the optimal static and automaton schemes are overlapping in Figure 2.7 means that one scheme cannot be assumed to be better than another. It can be seen that the learning automaton has converged close to the static optimum solution, whilst the dynamic threshold schemes' use of the queue state information further improves the average packet delay. For realistic network implementations, a static optimum result is the best we can hope for since gathering dynamic state (e.g. queue) information is rarely practical and optimal dynamic solutions only exist for very simple networks under simplified traffic assumptions. The delay inherent in gathering state information in practice means that the performance difference between the optimal static and dynamic state based schemes is further reduced. In Table 2.1 below, we compare the knowledge required by each of the three control approaches.

Control	λ	μ_1	μ_2	$q(1)$	$q(2)$	Delay Feedback
Static Optimum	yes	yes	yes	no	no	no
Learning/adaptive	no	no	no	no	no	yes
Dynamic threshold	yes	yes	yes	yes	yes	no

Table 2.1 - Comparison of knowledge required by controls

In practical network protocol implementations, the feedback required by the learning algorithm will usually come as a by-product of the operation of the protocol. This is the case since a user will generally expect some form of notification of success/failure from his/her request for service.

2.7. Summary

This chapter has reviewed the application of non-symbolic AI techniques to the control of communication networks. The specific AI techniques covered are Artificial Neural Networks (ANNs), Fuzzy Logic, Intelligent Agents and Stochastic Learning Automata (SLA). The benefits of a learning approach to a simple two path routing problem were investigated.

The application of ANNs and Fuzzy Logic has spanned several different areas of traffic control, many of the studies using artificially generated traffic models. Each application of AI tends to solve a different variant of the network control problem and the advantages over traditional approaches is not always apparent. There is a need to produce a more thorough comparison between AI based and traditional controls using benchmark problems, before AI will become truly accepted by the networking community.

It is thought that certain types of intelligent agent hold promise for the network control problem.

Specifically, 'reactive' agents were reviewed in some detail and a number of architectures were presented. There are few hard results available for such systems however and further experimentation is required, taking care to demonstrate the benefits that an agent system could bring to the problems which traditional techniques do not.

With learning automata, the benefits of their application are clear, that is, they provide an adaptive routing capability with very little available information. Indeed, without any access to the network topology or traffic demands, some form of learning/self-organising based routing may be the only viable approach for a lightweight adaptive routing. Although a future control structure may contain learning automata as one of the control layers (see section 2.4.5.), we feel that there is still a requirement to understand the operation of even a single layer. In subsequent Chapters therefore, we are interested in comparing learning automata for unicast and multicast routing in realistic network topologies with conventional techniques. As a first step, the next chapter considers the routing of traffic requiring multiple QoS constraints (e.g. bandwidth and delay), and examines the use of automata for routing in more complex environments, spanning issues such as aggregated routing, trunk reservation, large bandwidth-delay product networks and multi-rate traffic.

Chapter 3

Learning Automata for routing of 'real-time' traffic in Integrated Service Networks

3.1. Introduction

In this chapter, we briefly review the problem of routing 'real-time' (RT) traffic in integrated service networks, also known as 'Quality of Service' (QoS) routing. We explore the issues faced by QoS-based routing, explaining how a learning capability might help to address many of the complex issues involved. We explain the various ways in which automata may be implemented for real-time routing purposes. Finally, we present extensive simulation results for two network topologies comparing automata routing with existing techniques.

3.2. A brief review of QoS-based routing and related work

The current Internet does not support real-time flows and utilises shortest path algorithms to route best-effort traffic. Typical protocols are the distance vector based Routing Information Protocol (RIP) [92] and the link state based Open Shortest Path First (OSPF) [68], which are designed to operate within a single Autonomous System (AS), also known as Intra-Domain routing protocols. These protocols usually assign static metrics to links such that routing decisions will only adapt to changes in topology and not to changes in loading. The topology information is typically exchanged between all nodes periodically (e.g. every 5 secs – 30 mins [68]) so that nodes can react to changes in topology due to link/node failures. Thus, current Internet routing protocols provide resilience rather than any form of adaptive resource allocation. Once the Internet architecture can support some form of resource reservation, it is thought that Quality of Service (QoS) routing may be desirable. The idea of QoS-based routing is to select routes that have a high probability of meeting the QoS requirements of the requesting flow. In the simplest case, an incoming flow might only request the bandwidth required for the flow. In [93], the benefits of QoS based routing are stated as: (1) Dynamic determination of feasible paths; (2) Optimisation of resource usage; (3) Graceful performance degradation. In general, the aim of an adaptive resource allocation mechanism such as QoS-based routing is to provide reasonable performance (e.g. blocking probabilities) over a range of traffic and topology conditions, rather than being optimal for any particular scenario, since optimal techniques (if they are available) require knowledge of the traffic statistics. It is generally envisioned that

a QoS-routing capability would utilise some form of dynamic state (i.e. free resource) sensitive routing, which can help to compensate for inadequacies in network engineering made at the design stage. A typical QoS-based routing algorithm globally distributes topology, link resource availability, and (in some cases) per-flow resource usage [94]. Examples are presented in [95], [96], [97], [98], [99], [100] and good introductions to the problem are presented in [93] and [101]. In many respects, these proposals are similar to the dynamic routing strategies previously proposed for circuit switched networks (see [102] for an overview), although Integrated Services networks are likely to require new assumptions regarding topologies and traffic and thus require examination in their own right. Efficient algorithms have been proposed for dynamic routing in circuit switched networks such as 'Dynamic Alternative Routing' (DAR) [103] and 'Dynamic Non-hierarchical Routing' (DNHR) [104] although these algorithms have been primarily aimed at fully connected networks where the direct (one-hop) link is always chosen first, and the aim of a routing algorithm is to select the appropriate 2-link alternative if a call is blocked on the direct route. It is possible, for the fully connected case, to design efficient algorithms that store only local state since we never have to deal with routes more than 2-hops long. In the current Internet, we can commonly expect routes of at least 30 hops [105] and the design of efficient routing (and control) mechanisms based on local state becomes more difficult. In addition, for certain future services, it may be critical to achieve a successful set-up on the first pass in order to minimise the latency seen by applications. This is particularly true as bandwidth-delay products become very large and the propagation delay becomes the dominant component of end-to-end delays. In this thesis, we are therefore interested in algorithms that maximise the probability of a successful connection set-up in the first pass. It is well known that trying to pack many connections into the network through re-routing mechanisms such as 'crankback' [106] can lead to stability problems. Finally, for fully connected circuit switched networks, work of others has only considered bandwidth guarantees since the network topology and the routing protocol operation ensure that the source and destination nodes are a maximum of 2 links apart. We take the view that the combined traffic and topology are continually evolving and may not be completely known. We are interested in algorithms that meet multiple QoS-constraints, not just bandwidth (e.g. bandwidth and propagation delay). Even if we take the view that the topology is being designed to support QoS, adaptive algorithms will be useful if we have uncertain traffic conditions.

A typical dynamic state based scheme protocol is PNNI (Private Network-Network Interface) [107], a QoS-routing protocol designed in the context of ATM networks. PNNI is a link state protocol which propagates topology and resource availability information throughout the network using flooding. Possible mechanisms to reduce the amount of flooded information include hop-count limited floods, low frequency updates, quantisation of link state (e.g. utilisation) and tree based forwarding, although this means that QoS-routing decisions are now made with inaccurate or uncertain state information. (see, [100], [108], [109], [110], [111]). Despite these overhead reduction mechanisms, we feel that approaches to QoS-based routing like PNNI will not scale to large networks in the longer term. In fact, there are a range of QoS-based routing options spanning a range of dynamics that may offer substantially reduced

overheads. At one end, there are static algorithms such as shortest path routing, and at the other, there are extremely dynamic state based routing schemes using almost instantaneous state information like PNNI. In Figure 3.1 below, we show the range of routing options available:

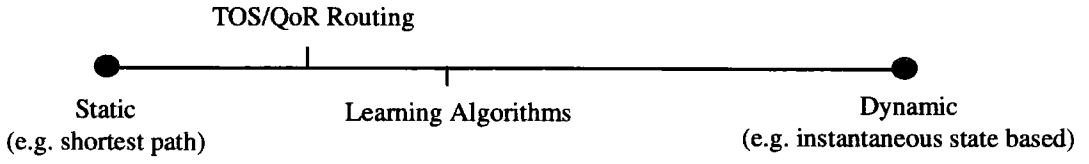


Figure 3.1 - Range of routing dynamics

Type-of-service (ToS) routing (also referred to as 'Quality of Route' (QoR) in [94]) has been suggested as a means of a more static type of QoS-based routing (see [21], [112]). Here, static service characteristics such as maximal bandwidth or propagation delay of a link are used to compute multiple paths to each destination. In terms of best-effort traffic for example, interactive services such as TELNET might be sent over low latency links whilst bulk transfers like FTP would be sent over the highest (static) capacity links. These metrics are envisioned to change fairly infrequently, alterations possibly being made by system administrators. The fact that static metrics are used rather than dynamic loading information means that many of the scalability problems of state sensitive routing are overcome. We have placed learning algorithms nearer to static routing since we believe that learning algorithms will provide an adaptive routing capability over fairly long timescales. We believe that the majority of dynamic QoS-based routing schemes will not be practical for the reasons explained below.

3.3. Advantages of Learning Algorithms for QoS-based routing

We believe that a learning scheme will have considerable advantages over the dynamic state based routing schemes discussed previously. In particular, learning automata for routing usually require no inter-nodal communication, requiring only a simple feedback signal regarding the success and/or failure of the connection set-up thereby minimising communication overheads. Additionally, we believe this feedback is likely to be contained in the operation of future protocols since users (applications) are likely to require some idea of how well they are performing. Thus, automata can use feedback signals which already exist in the operation of protocols. The majority of state based schemes being proposed require a separate flooding process to distribute the dynamic state information. We do not believe that such schemes will scale to very large networks, particularly at the inter-domain level, since flooding results in many unnecessary message transmissions and processing, and some domains may not allow access to certain state information. For short lived connections, particularly where the connection length is less than the propagation delay, one can question the purpose of gathering dynamic state information at all since the state in the network can change faster than we can measure. In this case, it is better to sample less frequently to get a picture of average longer-term resource availability in the network. Thus,

dynamic schemes start to appear more like static schemes. In [108], it has been found that dynamic routing schemes may be detrimental to throughput when considering very short lived connections. It is therefore proposed in [108] that one might utilise dynamic routing only for connections which are long lived, but keep static (e.g. shortest path) routing for short lived connections. If the traffic mix is dominated by short lived connections however, maintaining a dynamic routing capability may represent unnecessary overhead. Due to the expected dynamics of future networks, we feel that an adaptive or QoS-routing capability should operate on reasonably long timescales. In the case of learning algorithms, we have already seen that these algorithms tend to converge to the static optimum, which for extremely dynamic environments is possibly the best we can do. Furthermore, we expect the computational overhead (i.e. time to compute) to be extremely small for learning automata where we must simply choose from a (probably) small number of possible outcomes at random as well as performing a simple probability updating strategy. For shortest path calculations, the computation scales as $O(N^2)$ (Dijkastra), where N is the number of nodes in the network [90]. For the more complicated case of multicast routing, the computation for some heuristics can scale as $O(N^3)$ or $O(N^4)$ (see [113]), which can amount to a considerable overhead for large networks. If the computation times start to exceed the holding time of connections, we can question how efficient it is to use a QoS-based routing algorithm at all.

3.4. Stochastic Learning Automata (SLA) for QoS-based routing

Although there has been no work specifically examining the use of Learning Automata for QoS-routing, there exists some literature regarding the use of Learning Automata for the related problem of routing in circuit switched networks. The use of learning automata for adaptive routing was proposed by Narendra [78], in the context of telephone networks. Here, an action corresponds to a sequence of alternate paths to be attempted for call set-up, although an action can also correspond to a specific output link or path for the call to be routed on. Typical simulation studies ([77], [85] and [82]) have focused on fully connected circuit switched networks, and learning automata have been demonstrated to outperform fixed rule strategies, particularly under overload conditions. Also, there has been little consideration of the effects of delayed feedback in learning automata studies. This is an important consideration for high bandwidth-delay product networks since transportation lag is well known to cause oscillations in traditional control systems [114]. There is a need then to consider the performance of learning automata routing under the assumptions we expect for future integrated services networks. These include the need to consider more general topologies. In the Internet for example, a recent study has shown how Internet topology is considerably more irregular than traditional telephony networks [115]. Additionally, we expect real-time traffic to be 'multi-rate' meaning that an application requiring real-time service may request an arbitrary bandwidth rather than being constrained to a fixed bandwidth request as for circuit switched traffic. Previous learning automata studies have focused on traffic requiring simple bandwidth guarantees

whereas QoS-based routing may require selecting routes which meet multiple constraints (e.g. bandwidth and delay). Finally, we expect integrated services networks to support both real-time and non-real-time services and there is a need to study the interaction between the routing of both types of traffic.

3.5. Simulation Model

For the studies in this thesis, we used OPNET™, a discrete event simulator capable of simulating right down to individual packet events. However, for the real-time routing experiments reported here, we constructed a connection based or 'session level simulator' since the modelling of individual data packets is effectively redundant (from a routing perspective) once we have guaranteed a certain bandwidth to a connection. To validate the simulation models constructed, simple network scenarios were simulated. For example, a single source was simulated sending traffic to a single destination and the connection set-up packets were traced at each node in the path. It was then checked that connection attempts were being blocked correctly due to either lack of capacity, potential routing loop formation and delay bound violation. OPNET™ contains considerable functionality for debugging simulation models such as the ability to stop the simulation at arbitrary points in time, as well as printing out detailed information of all packet events and packet contents. The simulation models contained considerable 'anti-bugging' code to check for simple errors such as the available capacity at a node dropping below 0 for example. A more detailed guide to simulation model validation techniques is contained in [116]. We go on to describe the three main architectural considerations for the real-time simulator which were the topology, traffic generation and resource reservation models.

3.5.1. Topology

In this thesis, we provide simulation results for two network topologies. We present both of these in Figure 3.2 and Figure 3.3. The 10 node network is taken from [117] where it is proposed as a vehicle for dynamic routing studies. The 30 node network is taken from [118] and represents a hypothetical SDH (Synchronous Digital Hierarchy) transport network. Both networks were chosen such that there were a considerable number of alternative paths, and have an average nodal connectivity of between 3 and 4 reflecting Internet type connectivity levels. Although routing results are somewhat dependent on the topologies chosen, we felt that random generation of topologies would prove too computationally intensive and require many runs to gauge the average behaviour. Another option was to simulate uniform topology structures although we felt that these might prove rather unrealistic as compared to a global network such as the Internet. In Table 3.1, we summarise the important statistics of both networks. The topologies shown were specified as a set of nodes and bi-directional links in a configuration file to the simulator. Each link was modelled by a propagation plus processing delay that could be altered via the simulation tool. Each node was both a source and sink of traffic, as well as serving as a possible transit node for traffic between other nodes.

Topology Statistics	10 Node Network	30 Node Network
Average Node Degree	3.2	3.67
Max. Shortest Path Length (hops)	4	7
Average no. of Shortest Paths/source-destination pair	1.29	2.18
Average Length of Shortest Paths (hops)	2.12	4

Table 3.1 – Topology Statistics.

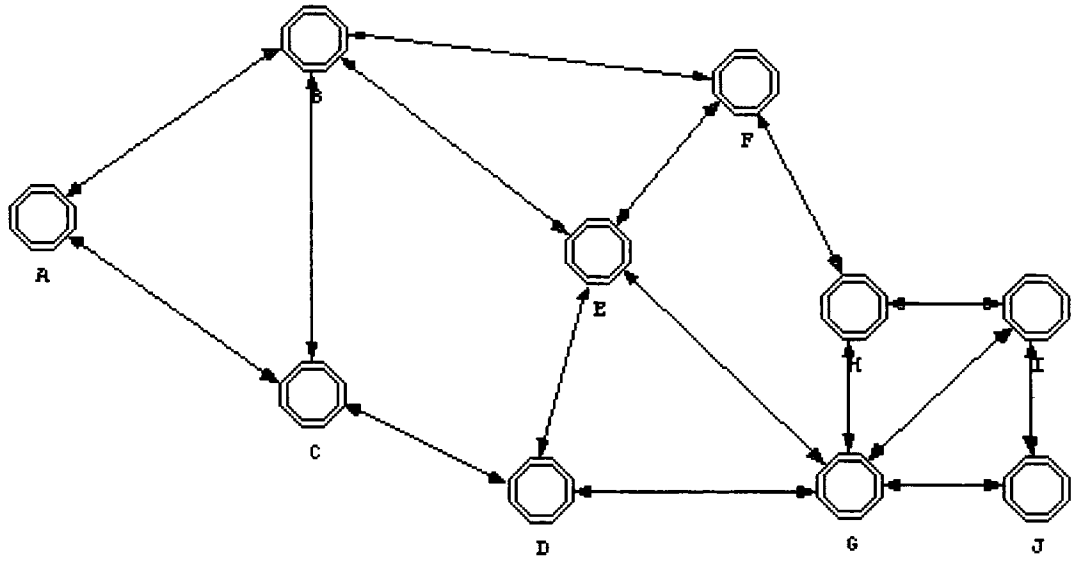


Figure 3.2 - 10 Node Network

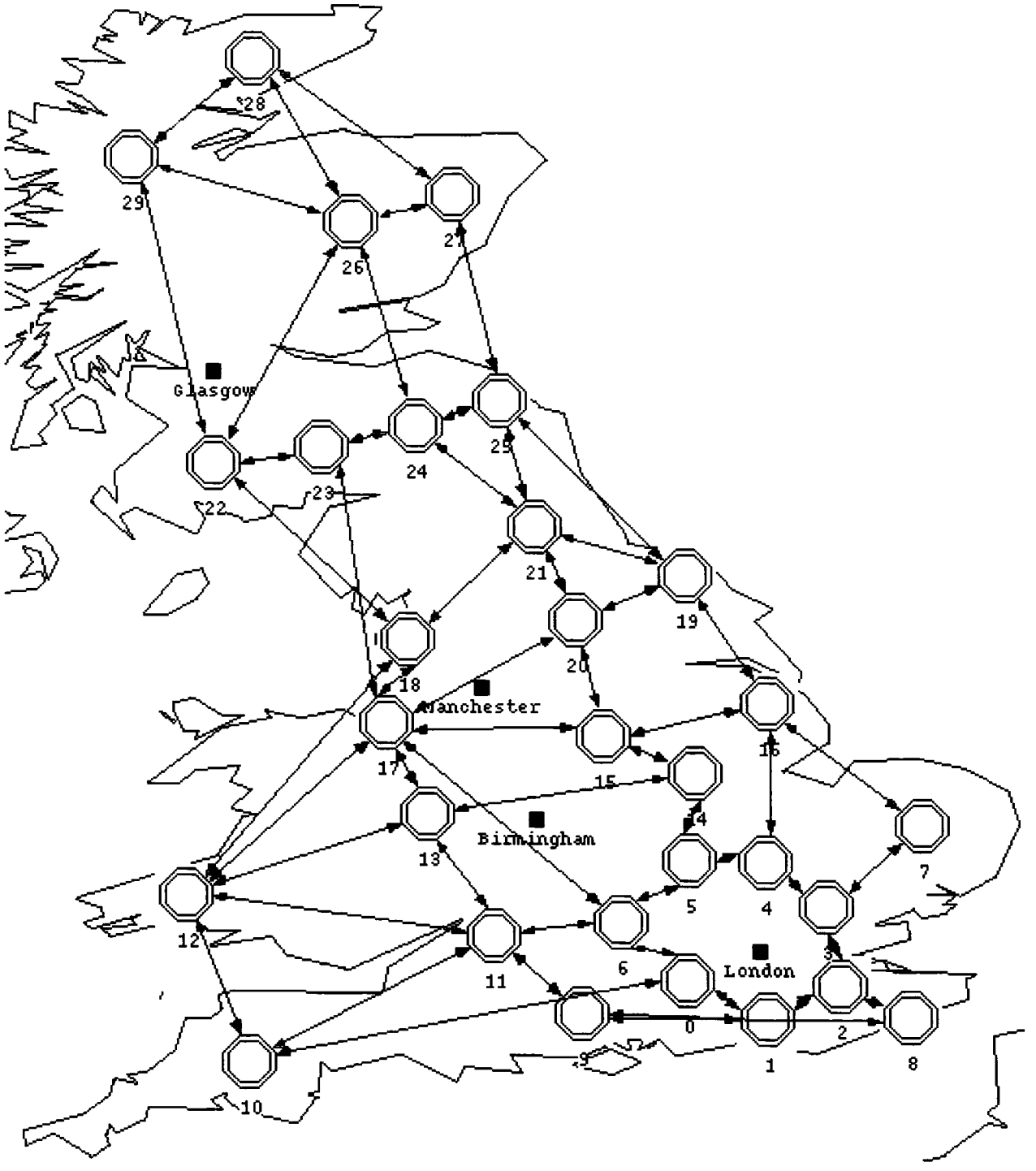


Figure 3.3 - 30 Node Network

The nodes contained the resource to be reserved for this study. This resource was simply defined as a number of available units at each node, which could be set in the topology configuration file. For the simulations reported here, this resource represented bandwidth to be reserved by user flows. We did not simulate the effect of link or node failures although it would be straight forward to modify the simulator to achieve this, and learning algorithms are capable of adapting to link/node failures as shown in [85]. As we have stated above, we believe that a QoS-based routing capability should aim to perform adaptive resource allocation over the long term, and is not likely to be used as a resilience mechanism. We feel that there are likely to be discrete control mechanisms operating on shorter timescales dealing with resilience issues in future integrated services networks. For example, a distributed restoration mechanism is investigated in [118].

3.5.2. Traffic Generation

In a similar manner to the topology information, the traffic parameters were read in via a configuration file to the simulator at simulation start time. It was also possible to read in alternative traffic configuration files during the simulation itself to simulate the effect of traffic changes. In generating the traffic, it was necessary to make assumptions regarding the QoS parameters likely to be required by real-time flows. For general QoS-based routing, to enable the selection of paths which meet QoS requirements, we must ensure that links and nodes are represented by the appropriate metrics. Metrics may include the residual bandwidth on a link, the propagation delay or even the reliability of a link for example. These metrics are important since they define the types of QoS guarantees the network can support [93]. The selection of metrics is also important from a computational perspective since some combinations of metrics prove to be extremely computationally complex. In [96], it is shown that the selection of paths to meet multiple QoS constraints is NP-complete for most combinations of metrics. A simple approximation that is commonly used is to find paths based on one metric (i.e. the primary metric) producing a reduced graph, from which, paths based on the secondary metric can be extracted. Examples based on bandwidth and path length are the 'shortest-widest' [96] and 'widest-shortest' [95] algorithms. We believe that end-to-end delay is the primary QoS parameter of interest to real-time flows and assuming a rate-based delay model, a source node can determine the end-to-end delay d (**p**) it expects to obtain from an n hop path **p** using the following expression [110]:

$$d(\mathbf{p}) = \frac{\sigma + cn}{r} + \sum_{l \in \mathbf{p}} d_l , \qquad (3.1.)$$

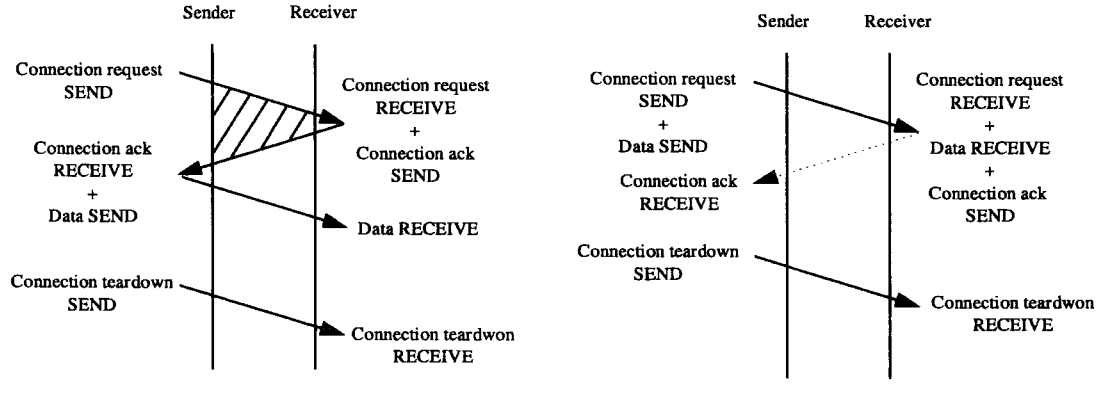
where σ is the size of the connection's burst and r the connection's minimal rate according to the leaky bucket model, c is the connection's maximum packet size and d_l is the propagation delay at link l. Thus, if we have knowledge of the likely range of lengths of a path routed between a particular source-destination pair, the source can calculate the range of bandwidths required, r, to meet an end-to-end delay bound d (**p**). It is up to the source how it calculates the bandwidth required. For example, the bandwidth

r could be calculated according to some 'equivalent bandwidth' definition [119], or more simply through peak-rate allocation (i.e. burst size σ is zero). The network will simply see a bandwidth or rate r to be guaranteed however the source decides to calculate this value. Ideally, a routing algorithm will select routes that are within one or two hops greater than the shortest path length so that sources can request a bandwidth r with some accuracy. In this thesis, we assume that user applications will require a guaranteed bandwidth and a path length (or alternatively propagation delay) guaranteed to be within a certain number of hops of the shortest path length for the source-destination pair in question. We believe that a guaranteed bandwidth and propagation delay constitute a practically achievable QoS set, from which, other QoS parameters such as the delay variation can be derived. To monitor whether a connection request had not violated any path length bound, the reservation set-up packet contained a field which recorded the number of hops that the packet had travelled as well as the number of hops allowable. This is similar to the 'time-to-live' (TTL) field in IP packets [120] which is decremented at each hop and designed to prevent packets staying within the network indefinitely after a routing loop has occurred.

The traffic file read in at simulation start up contained the following information. Firstly, the traffic arrival rates (λ), for each source-destination pair, and mean session holding time ($1/\mu$), were read in. The bandwidth required by each flow was also specified in this file together with any desired path length constraint. It was a simple matter to amend this file for multi-rate traffic by extending the file to include all of the above information for however many traffic rates (bandwidths required) were needed. In the experiments performed, session inter-arrival times (1/arrival rate) and session holding times were exponentially distributed unless otherwise specified.

3.5.3. Resource Reservation Mechanisms

For the simulation of real-time routing algorithms, we considered two signalling methods that might be used to reserve resources in the network. We describe both of these methods with reference to Figure 3.4 below.



(a) Traditional Set-up **(b) 'On-the-fly' Set-up**

Figure 3.4 - Two possible set-up mechanisms.

The signalling mechanism shown in Figure 3.4 (a) is the type of mechanism used for interactive services such as telephony (also ATM). Here, the source sends a request to the receiver, routed by the real-time routing algorithm, and installs reservation state in the intermediate nodes, not sending data unless it receives a connection acknowledgement indicating that a real-time connection has been successfully set up. If the connection request fails at any point from source to destination, a connect 'reject' signal is sent back to the source removing the reservation state at intermediate nodes. In the simulations reported here, a connection could be rejected at a node if there was insufficient capacity at the node, if the request violated the path length bound requested or if there was a routing loop formation. The second signalling mechanism which we refer to as 'on-the-fly' based signalling, does not require the explicit set-up phase, sending the data immediately after the connection request (i.e. first data packet contains reservation request). This 'On-the-fly' signalling has the advantage that we do not have to wait for the traditional set-up phase before we start to send user data. This may be important for networks with high-bandwidth delay products since the set-up phase effectively represents a delay over which we are denied access to the network bandwidth (shaded area in Figure 3.4 (a)). We envisage that this reservation mechanism would prove useful for non-interactive services where we still require real-time service. (e.g. guaranteed data rate for best-effort traffic). For the on-the-fly-signalling model, we have looked at two variations. If the connection request fails at a node, the connection reject signal may or may not remove reservation state in the intermediate nodes. For example, we might choose to forward the data under another service class (e.g. best-effort) and not have the connect reject signal remove the reservation state at the previous nodes. This is shown in Figure 3.5. It is essentially up to the user application which failure mechanism should be used. In summary, we have defined three reservation mechanisms which have been used in simulations. We summarise these in Table 3.2. The traditional signalling mechanism is used in all subsequent results until section 3.7.10.

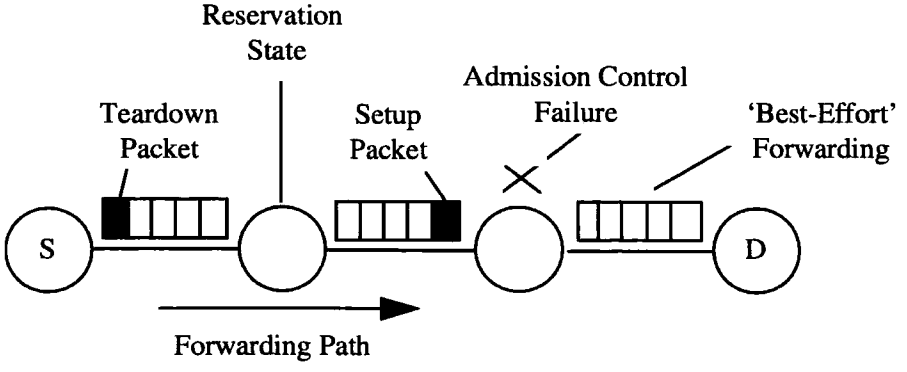


Figure 3.5 - 'On-the-fly' signalling model, with best-effort forwarding of data after admission failure.

	Set-up Phase	Connection Reject Removes Reservation State
Traditional Signalling	Yes	Yes
'On-the-fly' 1	No	Yes
'On-the-fly' 2	No	No

Table 3.2 - Summary of three reservation mechanisms

3.6. Operation of Stochastic Learning Automata (SLA) for QoS-based routing

The basic theory regarding Stochastic Learning Automata (which we also refer to simply as Learning Automata) is contained within Appendix A. Here, we explain how learning automata can be used for the routing of real-time connections. We describe two ways in which automata have been applied, for source routing and hop-by-hop routing.

3.6.1. Source Routing

Here, each potential source node in the network is assumed to store a number of paths to each destination. Thus, different flows or sessions at a source with the same destination address may follow different paths within the network. A working group within the IETF has developed SDR (Source Demand Routing) [121], which enables the source-initiated selection of routes, designed to complement existing intra and inter-domain routing protocols. Learning automata for source routing can be considered as an approximate model to user/application behaviour if routing propagates to the edge of the network. The advantage of source routing is that it avoids routing loops and sources may use different algorithms to select the source routes. To meet propagation delay bounds, the source can choose alternate paths with a bounded length. For example, the source may choose to select routes which are within 2 hops of the shortest path length for the particular source-destination pair in question. In this way, the source can calculate the bandwidth required to meet the user application delay requirements. The problem with source routing is that we require sufficient topological information so that we can construct paths for each source-destination pair. The source nodes must also have sufficient storage to store the complete source routes. Although this may be possible within a single domain, it is more difficult to achieve at the inter-domain level, where the aggregation of information and possible policy constraints may prevent us creating a complete source-destination route. In [94], source nodes interrogate other nodes in the network to build up a partial map of the network and calculate source routes. For a network whose topology is not changing that dynamically, caching may be useful here to reduce the need for such interrogations. In the simulations performed here, each source node stored a certain number of complete paths to each

destination. Also, each source node contained a learning automaton for every other destination node. For an N node network therefore, there will be a total of N(N-1) learning automata, each node containing N-1 automata. The probability vector of each automaton contains the probability of using a particular source route to the destination. Once a source route has been selected, the source route is installed in the set-up packet which follows the given route to the destination. If each source node stores k paths to each destination, each automaton will have a probability vector containing k probabilities. Thus, if the average length of a stored source route is L, the amount of information stored at a single node will scale as O([kL+k]N), whereas the information stored will scale as O(N) for a simple distributed shortest path scheme. To select source routes to store, we first chose from the set of shortest paths, then shortest paths plus one hop, then plus two etc.. until we achieved the required number of source routes. The more source routes we store, the greater selection automata have to choose optimal routes at the expense of increased storage requirement. We also have more flexibility in being able to route around localised congestion build-ups. Storing more routes means however that automata will require more attempts to reach steady state convergence since each automaton has more actions. For the updating process of the automata probabilities, all of the resource reservation mechanisms discussed previously assume that the source receives an indication of the success or failure of the connection set-up. For source routing, only the source probabilities were updated using the binary feedback signal in conjunction with the particular reinforcement learning algorithm adopted. Automata probabilities were initialised to 1/(number of source routes). (i.e. equal probability of each source route).

3.6.2. Hop-by-hop Routing

The second implementation of learning automata represents a more distributed approach to the adaptive routing problem. There are a number of options regarding the granularity of routing decisions. In the current Internet for example, nodes route incoming packets based only on the destination address contained within a field in the packet. As pointed out in [93], routing entirely based on destination address typically means that all flows between any source and the destination will be routed over the same paths which limits flexibility if this path is congested. One step up in granularity is to route sessions based on source and destination address although all flows between a specific source-destination pair will be routed over the same path again limiting flexibility. For ultimate granularity, we can route based on individual flows or sessions generated between a particular source-destination pair although this will result in a large flow state storage requirement within the network. With learning automata hop-by-hop routing, each node typically routes sessions based on the destination address. Thus, for an N node network, each node stores N-1 automata, one for each destination. The probability vector of each automaton represents the probability of forwarding a connection request on a particular link. The set-up requests were sent out on a link different to the one that they came in on, to prevent the formation of trivial routing loops. In previous studies, the links on which a set-up request is routed on can be limited to a feasible set, if we have knowledge of the global topology [85]. When the set-up packets are not

constrained to choose particular output links (other than the link the set-up packet came in on), this routing has been referred to as 'self-organising' routing [85]. In this thesis, all subsequent hop-by-hop automata implementations are of the self-organising type, since we are interested in those adaptive mechanisms that may operate with the minimum of information. It is unlikely for example that we would have knowledge of the topology of remote domains, necessary to constrain the automata to feasible decisions. Although we route the connection set-up based on destination address, since the routing process is probabilistic, there is a chance that flows between a particular source-destination pair may be routed over different paths and we must either maintain flow specific state information telling us the next hop to send packets to for a particular flow, or the data packets themselves must specify the complete source-destination route. The routing process is depicted in Figure 3.6 where the automaton at node m for destination j, A_j^m, routes an incoming reservation request to node m+1.

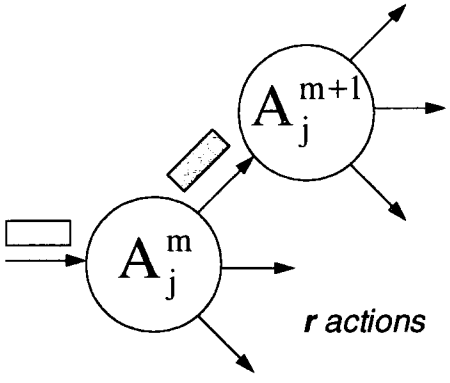


Figure 3.6 - Automata routing action.

For a typical shortest path approach, the storage requirement of a node's routing table scales as O(N) since each node stores a next hop value for each destination node. For automata, the equivalent storage requirement approximately scales as O(cN) where c is the average node degree of the network, since each node now stores a probability value for each link for each destination node. For additional granularity, we may choose automata to route based on source and destination address where each node now stores (N(N-1)-(N-1)) automata, information storage now scaling as $O(cN^2)$. This may increase steady state performance at the cost of increased convergence times and nodal storage requirement. For the updating of the automata probabilities, the connection accept/reject packet backtracked through the chosen route updating the intermediate automata probabilities using the particular reinforcement learning algorithm adopted. The backtrack process was possible since set-up packets stored the address of nodes visited. Automata probabilities were initialised to 1/(number of links) (i.e. equal probability of any output link).

3.7. Results

3.7.1. Even Traffic Demands

For the initial experiments, the traffic demand followed an even distribution whereby all source-destination pairs generated connection requests at the same rate (see Appendix C). Nodal capacities were set to 50 and propagation (+processing) delays were set to 1 millisecond. Incoming calls consumed a single unit of resource and had exponential holding times of 30s. Interarrival times were also drawn from an exponential distribution. To measure performance for real-time routing studies, network throughput or blocking probability is usually recorded and was also adopted in this thesis. In Figure 3.7, we plot blocking probability against nodal arrival rate for the 10 node network in Figure 3.2 for source routing automata, hop-by-hop automata, shortest path and random routing. For the source routing automata, each source stored two paths to each destination. For hop-by-hop automata, connection set-up packets could travel a maximum of 10 hops. Random routing can be considered to be hop-by-hop automata routing with learning rates set to 0. Finally, the LRI and LRP reinforcement algorithms (see Appendix A) were employed with a learning rate, a = 0.03 (b = 0/0.03), which was determined after a tuning process. A lower bound on the performance of any routing algorithm can be plotted using Erlang's formula (see Appendix B) and is also shown in Figure 3.7 together with the previous algorithms.

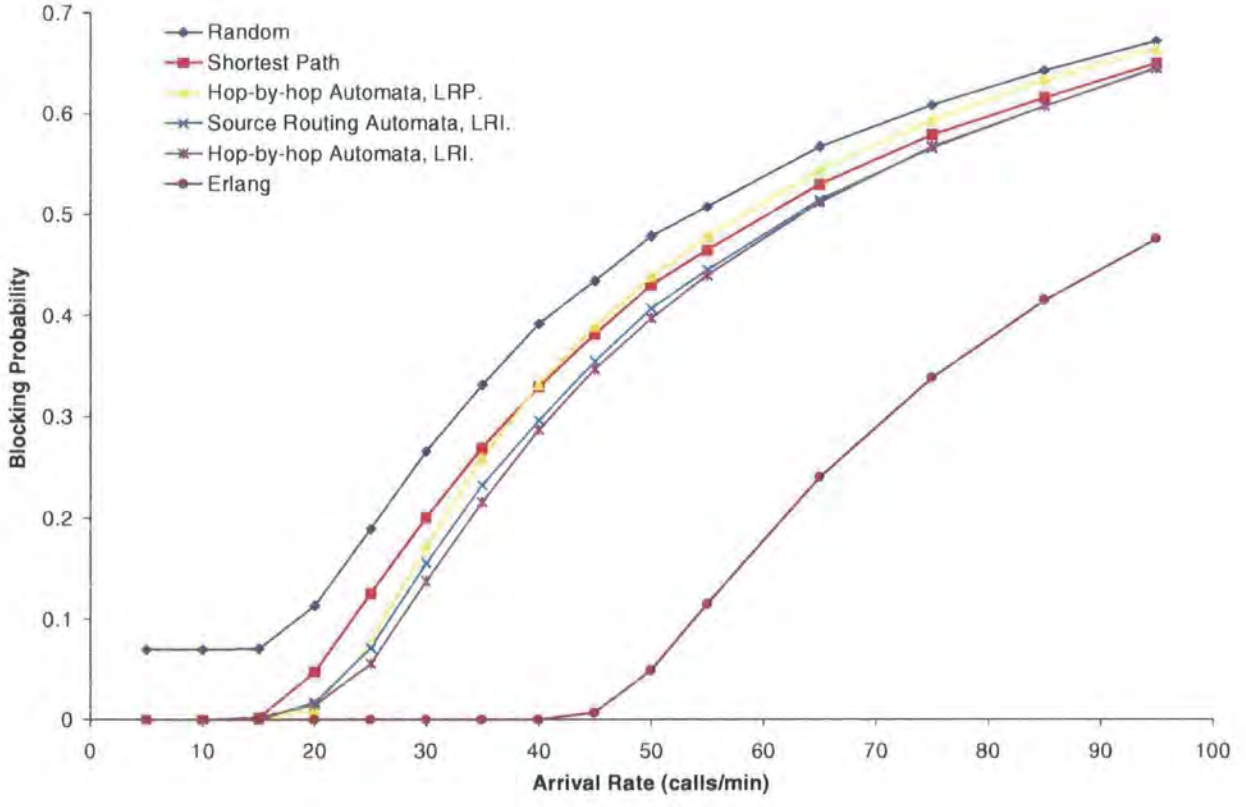


Figure 3.7 – Steady state blocking probability, 10 node network, even traffic.

To measure steady state blocking probability for all experiments in this thesis, an appropriate number of connection attempts were ignored for transient removal purposes. This window varied according to the number of connections needed for the automata to converge, which was measured using transient traces of blocking probability and entropy (see Appendix A). In addition to long simulation runs, a number of seeds were simulated for each point to give sufficiently small confidence intervals, which have subsequently been removed for clarity. Looking at Figure 3.7, it can be seen that both learning automata schemes provide consistently lower blocking probability over the range of nodal arrival rates. At very low loads, adaptive routing provides little advantage over a shortest path scheme since there is little congestion within the network and no benefit in taking alternate routes. Similarly, at very high loadings, calls are likely to be blocked whatever path is chosen due to high levels of congestion. The low-mid arrival rate range provides the maximum benefit of learning automata routing over a shortest path approach through the automata schemes' use of alternate paths. We also note that the difference between the hop-by-hop and source routing automata approaches is rather small. This shows that reasonable use of alternate routing can be achieved when only a few pre-computed paths are stored by the source nodes. The LRP hop-by-hop automata can be seen to produce higher blocking probabilities than the LRI based hop-by-hop automata. In stationary environments, the behaviour of the LRP automata is ergodic whilst the LRI automata are ε -optimal, such that the LRI automata may converge to the optimal action within some tolerance bound (see Appendix A).

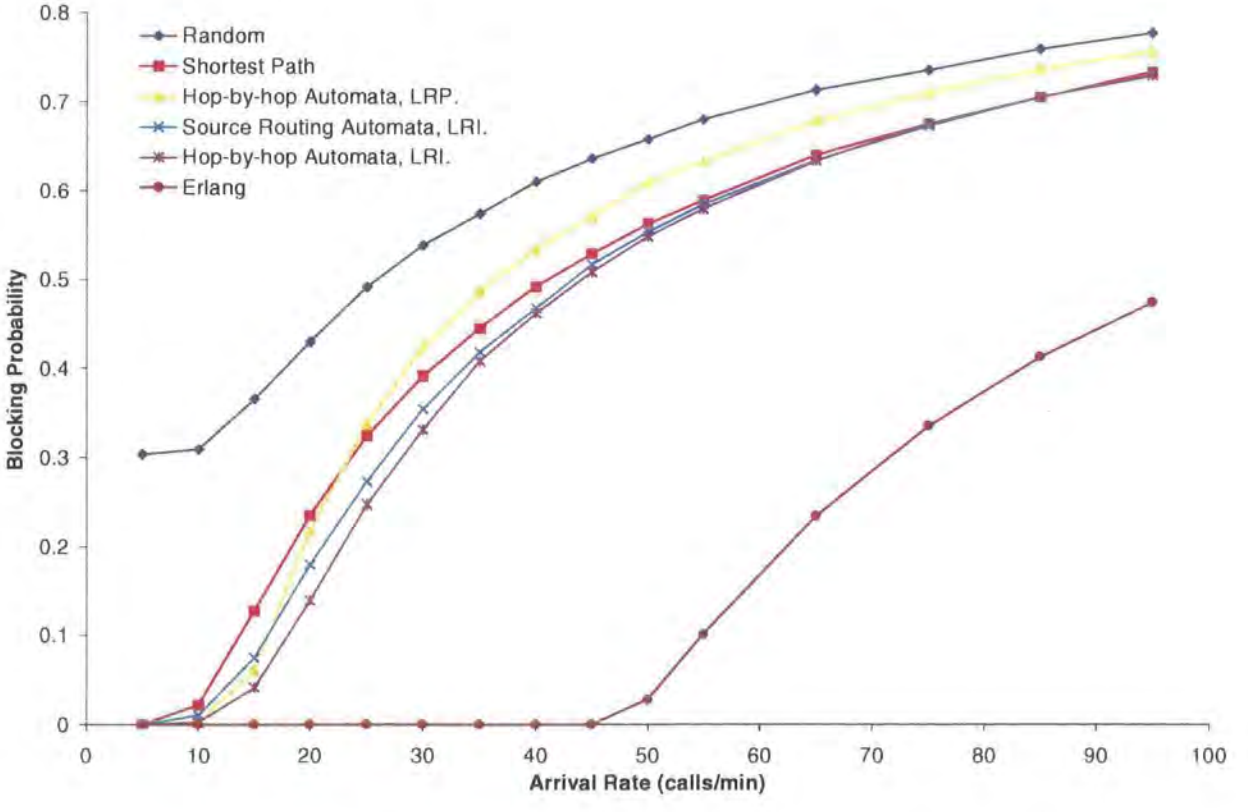


Figure 3.8 – Steady state blocking probability, 30 node network, even traffic.

In the (non-stationary) networks studied here, this means that the LRI automata are much more likely to reach the static optimum solution. In practice, the LReP reinforcement with a large reward and a small relative penalty value would be the preferred option since it is both ergodic and ε-optimal in stationary environments. In Figure 3.8, we plot the same curves for the 30-node network in Figure 3.3, the only difference being that reservation requests could now travel up to a maximum of 13 hops. It can be seen that the hop-by-hop automata still provide reasonable improvement over a shortest path approach, the source routing automata providing a good approximation.

3.7.2. Uneven Traffic Demands

In the next set of experiments, the capacity of node E in the 10 node network, Figure 3.2, was reduced to 5 while keeping all other nodes at 50. In addition, the traffic matrix was configured so that a significant proportion of shortest path traffic should go through node E in order that it should become easily overloaded (see Appendix C). In this way, we demonstrated the benefits of automata based routing when a particular resource became overloaded in addition to uneven traffic demands. Steady state blocking probability for random, shortest path, hop-by-hop and source routing automata are plotted in Figure 3.9. The automata schemes used the LRI reinforcement algorithm with learning rate, a = 0.03 as before, and connection requests could travel a maximum of 10 hops.

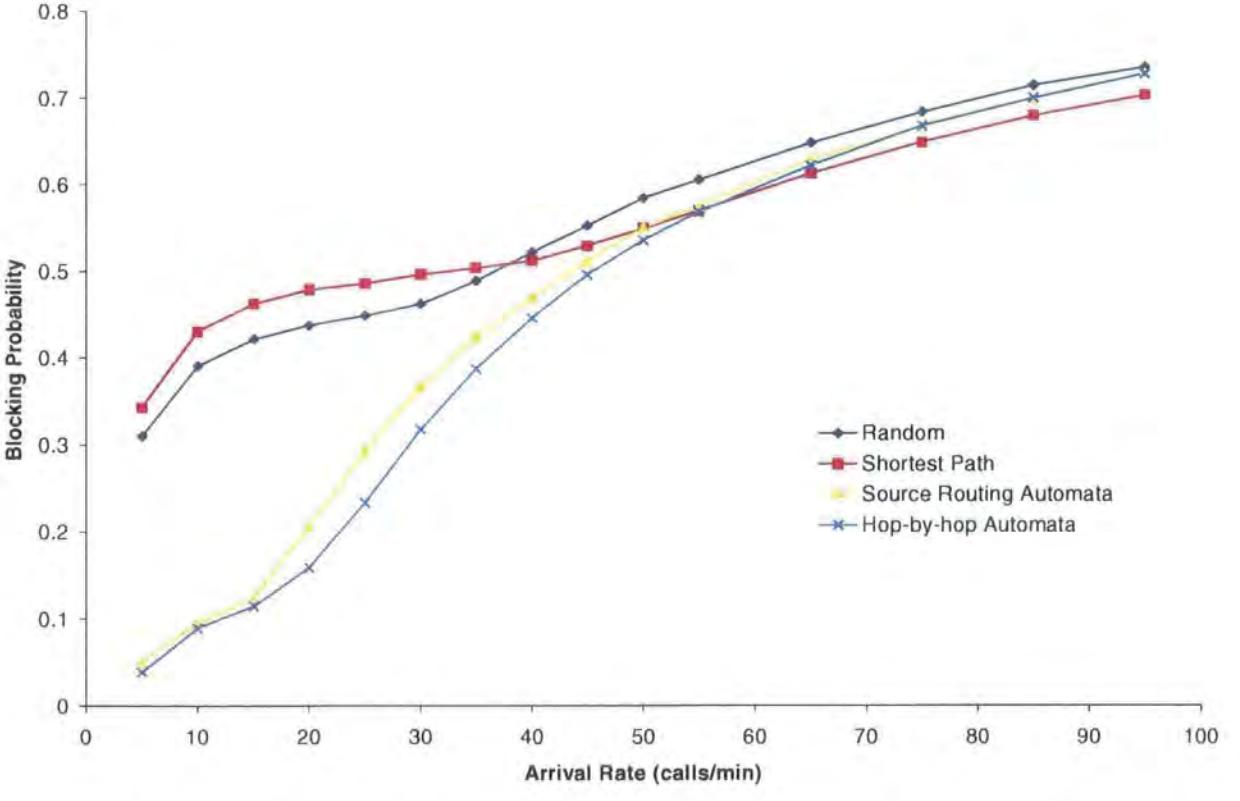
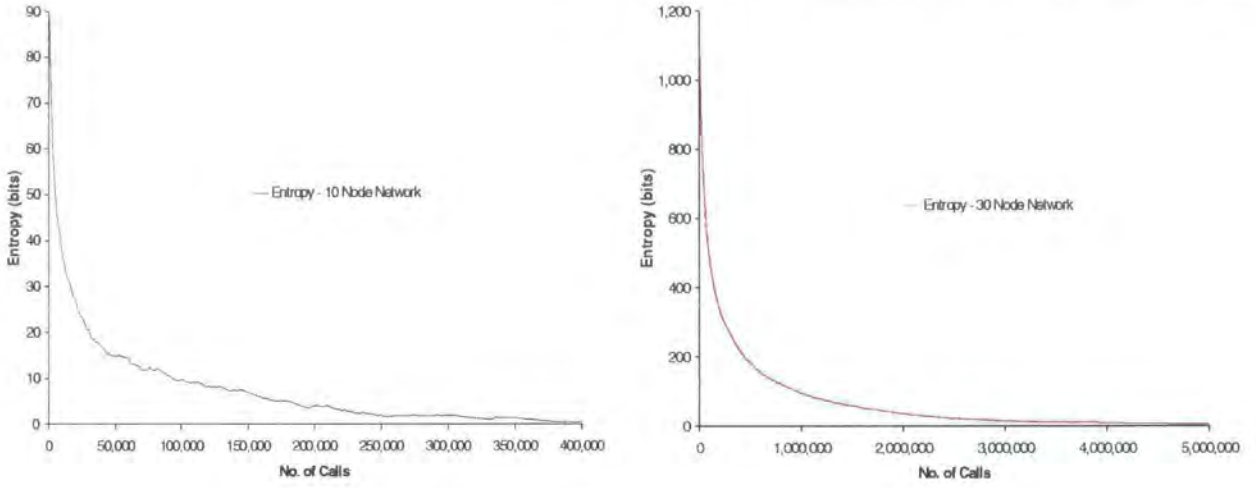


Figure 3.9 - Steady state blocking probability, 10 node network, uneven traffic.

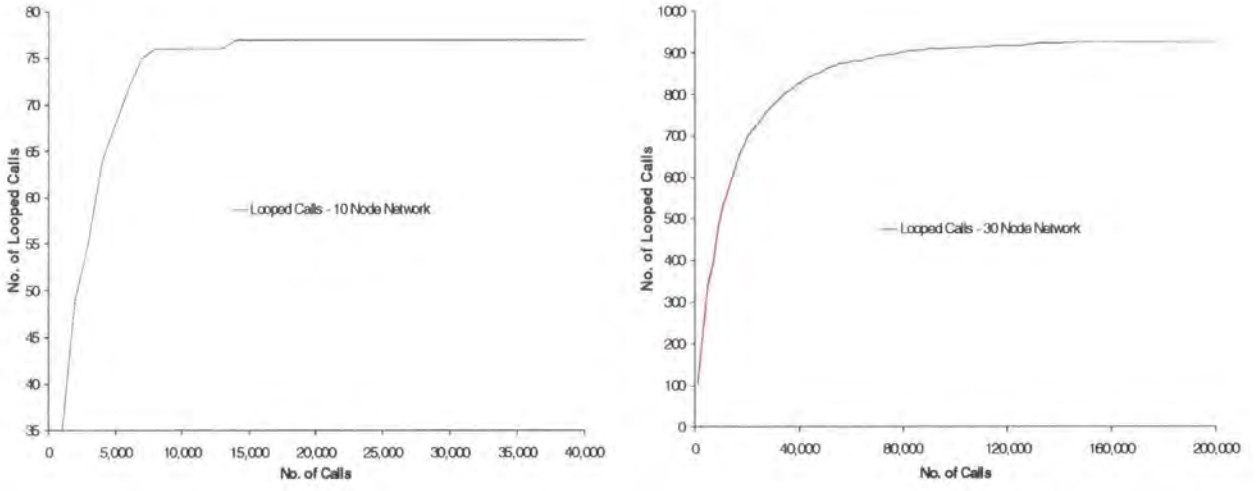
It can be observed that the use of alternate paths is critical in such traffic and topological scenarios, leading to around 40 percent difference in blocking probability in this example between shortest path and automata based routing schemes. In addition, we again note the source routing automata provide a close match to the distributed hop-by-hop automata despite storing only two paths per source-destination pair. We believe that localised resource shortages similar to that examined here could easily occur in future integrated services networks due to the sheer variability and uncertainty of incoming traffic demands, even for networks which are overprovisoned (for 'average' usage) to a large extent. Since we expect these networks to support different classes or priorities of traffic, the variance in resource availability could be significant for those lower priority traffics which have access to the 'left over' bandwidth from the higher priority traffics.

3.7.3. Convergence

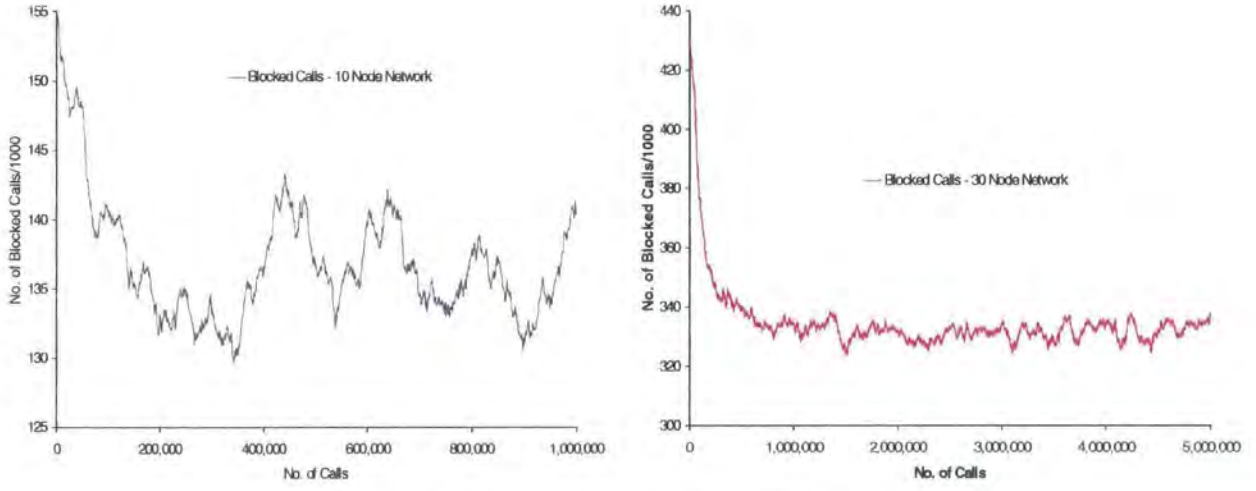
To display the convergence of the self-organising system of automata, experiments measuring the number of (average)blocked requests, entropy (see Appendix A) and looped connections were performed for the 10 and 30 node networks under even traffic demands. Sample paths are shown in Figure 3.10. The nodal arrival rate, λ, was set at 30 connections/minute and automata used LRI reinforcement with a learning rate, a = 0.03. The looped connections statistic measured the cumulative number of connections blocked due to the formation of routing loops. All statistics were measured every 1000 connection attempts. We see that the rate of change of number of connections blocked due to looping decreases as the automata learn to select feasible routes within the network. Similarly, the entropy plots show how the network of learning automata become more organised as the number of call attempts increases. Interestingly, the number of blocked connection requests seems to level out long before the entropy trace, suggesting that the initial convergence period is most influential to the performance of the learning automata and that subsequent connection attempts bring diminishing returns. From the previous analysis, we know that the total number of (destination address based) automata within a network scales as $O(N^2)$ and that the total number of actions will scale as $O(N^2)$ assuming that the nodal connectivity remains constant. From this simple analysis, we would expect the number of connection attempts to converge for the 30 node network to be $(30/10)^2 \approx 10$ times greater than for the 10 node network. This however ignores the interaction /interference between learning automata which may extend convergence. Looking at the transient traces, we see that the plots seem to roughly indicate a factor of 10 increase in convergence for this example.



(a) Entropy, 10 and 30 node networks.



(b) Looped Connections, 10 and 30 node networks.



(c) Average Blocked Requests, 10 and 30 node networks.

Figure 3.10 - Transient Measurements, 10 and 30 node networks.

3.7.4. Trunk Reservation

We note from the blocking probability traces in Figure 3.9 that at high loadings, shortest path routing gives lower blocking probability (or higher throughputs) than the automata based routing schemes. This is a result that has been investigated for dynamic routing in circuit switched networks (see [103], [122]). In these networks, it was found that routing along longer (2-hop) alternate paths 'steals' resource from the directly routed (1-hop) paths, increasing overall levels of blocking at high network loadings. It was also found that dynamic routing could exhibit bistable or oscillatory behaviour in link loadings at certain critical traffic levels where the link level loading could jump from high to low levels or vica versa. (see [103]). To solve these problems, some form of threshold policy (as in Chapter 2) or *trunk reservation* type scheme is used. The idea is to only allow calls (or connections) to be routed over longer alternate paths if the level of directly routed traffic is below some threshold. The use of such thresholds effectively introduces hysteresis into the system, helping to prevent oscillations as well as increasing throughputs at high loadings, although it can decrease throughputs at lower loadings as the number of alternate routing options is effectively decreased. Routing studies have shown that the optimum threshold is dependent on many factors including topology and the traffic loading [123]. In Figure 3.9, we see that the cross-over point between shortest path and automata routing occurs at a blocking level of around 50%. We believe that automata should be designed to operate over fairly long timescales such that the learning rates used will be reasonably low thus avoiding any danger of oscillation. However, we would still prefer automata to choose alternate paths that are within a few hops of the shortest paths since as well as increasing throughputs at high network loadings, we are able to give a practical path length bound to applications requiring bounded end-to-end delays. To achieve this, we required that each node knew the shortest path distance to each destination (in hops). When a source generated a join request, it placed the hop count requirement in the set-up packet. The hop count requirement could be 0 or more hops greater than the shortest path length. In this way, we ensured that the learning automata converged to reasonably short paths as dictated by the hop-count bound. As the shortest path length of a source-destination pair may be costly to acquire in practice (e.g. Dijkastra), one simple method is for each node to store an estimate of the shortest path length for each source-destination pair. Since reservation set-up packets are assumed to store the route travelled, a source can calculate the length of the path taken for a successful set-up upon receiving the connection accept signal. The estimate of the shortest path length for the source-destination pair in question can then be updated if the path just taken is less than our current estimate. Since we expect learning automata to utilise shortest paths fairly frequently, the set of distributed nodes will quickly build up knowledge of the shortest path lengths for all source-destination pairs. If the topology changes, we will need to re-initialise these estimates to ensure feasible hop-count requests. Since we expect network control interaction at many timescales (see Chapter 2), we envisage that a network resilience layer would send a signal to the (adaptive) routing layer informing it of a topology change such that the nodes would re-initialise their estimates of the shortest path lengths in the network. In Figure

3.11, we present a bar chart showing the relative path length of successful set-ups in the steady state for the case where a connection set-up may travel a maximum of 10 hops (i.e. unconstrained automata routing) for the 10 node network. The measurement window was 25,000 total call attempts and the nodal arrival rate was set at 30 connections per minute for the even traffic distribution.

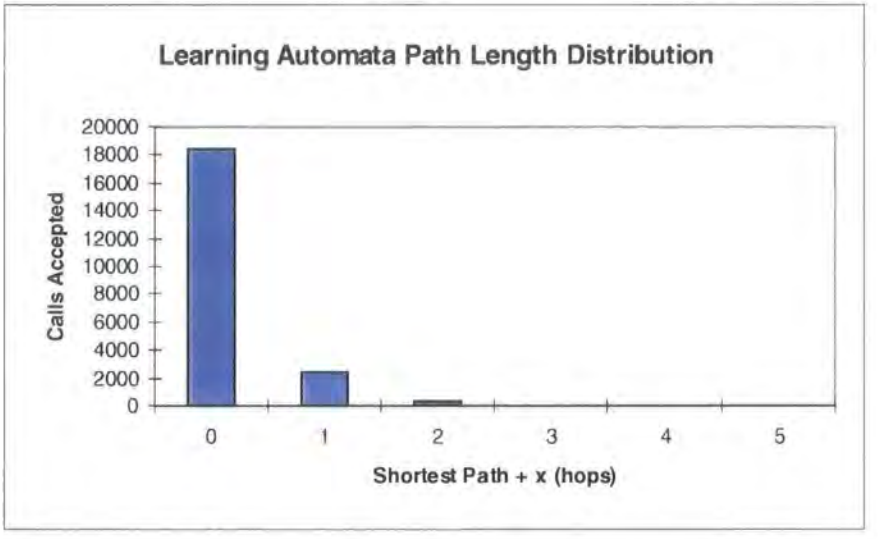


Figure 3.11 - Automata Path Length Distribution

We see from Figure 3.11 that even for effectively unconstrained routing, learning automata route the majority of connection requests over the shortest paths. In this case, 87% of calls accepted are on shortest paths and approximately 11% are accepted on a path one hop longer than the shortest paths. The remaining 2% of accepted connections are distributed over slightly longer paths. Thus, the unconstrained automata tend to naturally converge to the use of shorter paths. Constraining automata to shorter paths still, via the hop-count request bound, ensures that connections are never routed over excessively long paths ensuring a low end-to-end delay to applications. For the 30 node network, we have investigated the effect of changing the hop-count bound. In Figure 3.12 and Figure 3.13, we plot blocking probability and mean path length (of successful set-ups) against arrival rate for automata with a hop count bound of 13 hops (unconstrained) and a hop count of 0 hops greater than the shortest path length (constrained). Also plotted for reference are the shortest path and random routing schemes. Points in Figure 3.13 are plotted with 90% confidence intervals. We see in Figure 3.12, that the automata with the 0-hop constraint produce higher blocking at low loadings since we are unable to make use of load splitting among as many alternate paths, although they produce lower blocking at high loads since biasing towards shorter paths means that we interfere less with directly routed traffic. It can be observed from Figure 3.13 that automata based routing schemes favour the shorter paths with sensible use of alternate paths, as reflected in the closeness of the automata mean path length to that of the shortest paths. The mean path length for all traces decreases with increasing loading since connections requiring shorter paths have higher probability of being accepted than connections using longer paths.

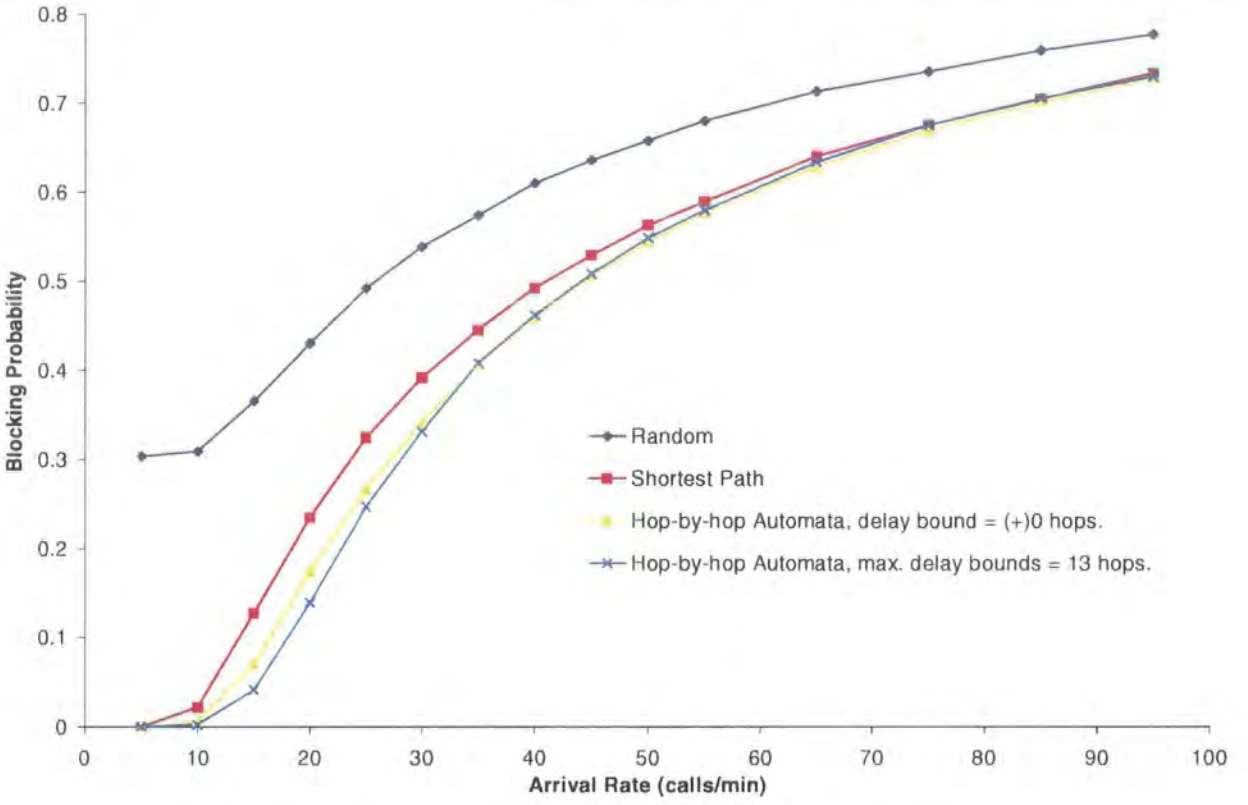


Figure 3.12 – Blocking Probability for different hop-count bounds

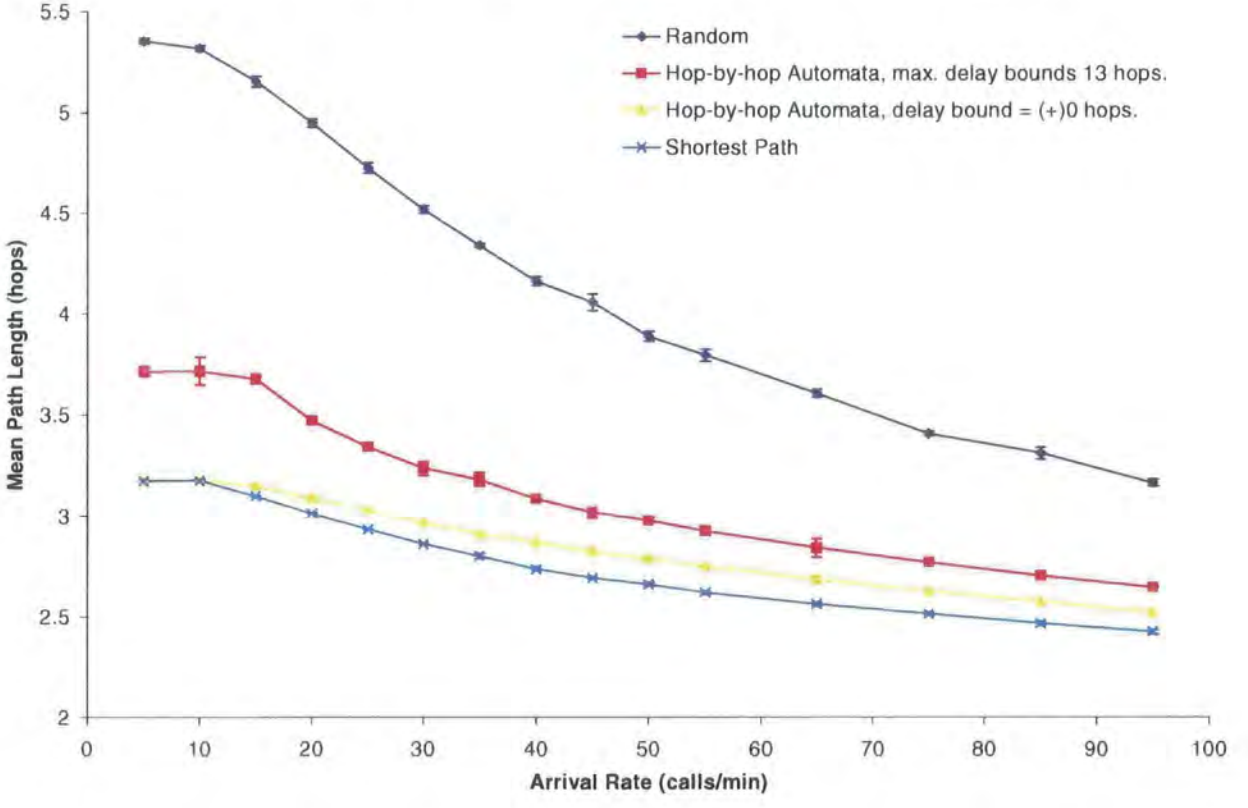


Figure 3.13 – Mean Path Length (Hops) for different routing schemes

3.7.5. Granularity of Routing Decision

As explained earlier, choosing the correct granularity (i.e. destination, source-destination or flow based) is necessary to achieve the target trade-off between performance and storage overheads. In Figure 3.14, we plot steady state blocking probability against arrival rate for learning automata schemes based on destination and source-destination granularities. For these experiments, even traffic demands were adopted for the 10 node network, and all automata used LRI reinforcement with a learning rate, a = 0.03. We see that the extra routing granularity of the source-destination based automata gives little improvement in blocking probabilities over the destination based automata for this scenario, indicating that sufficient load balancing can be achieved using automata based on destination address only. Although dependent on the topology and traffic demands, we suspect that destination based automata will be adequate for load balancing and avoiding localised congestion in future networks, helping to minimise the storage overhead of a typical automata implementation.

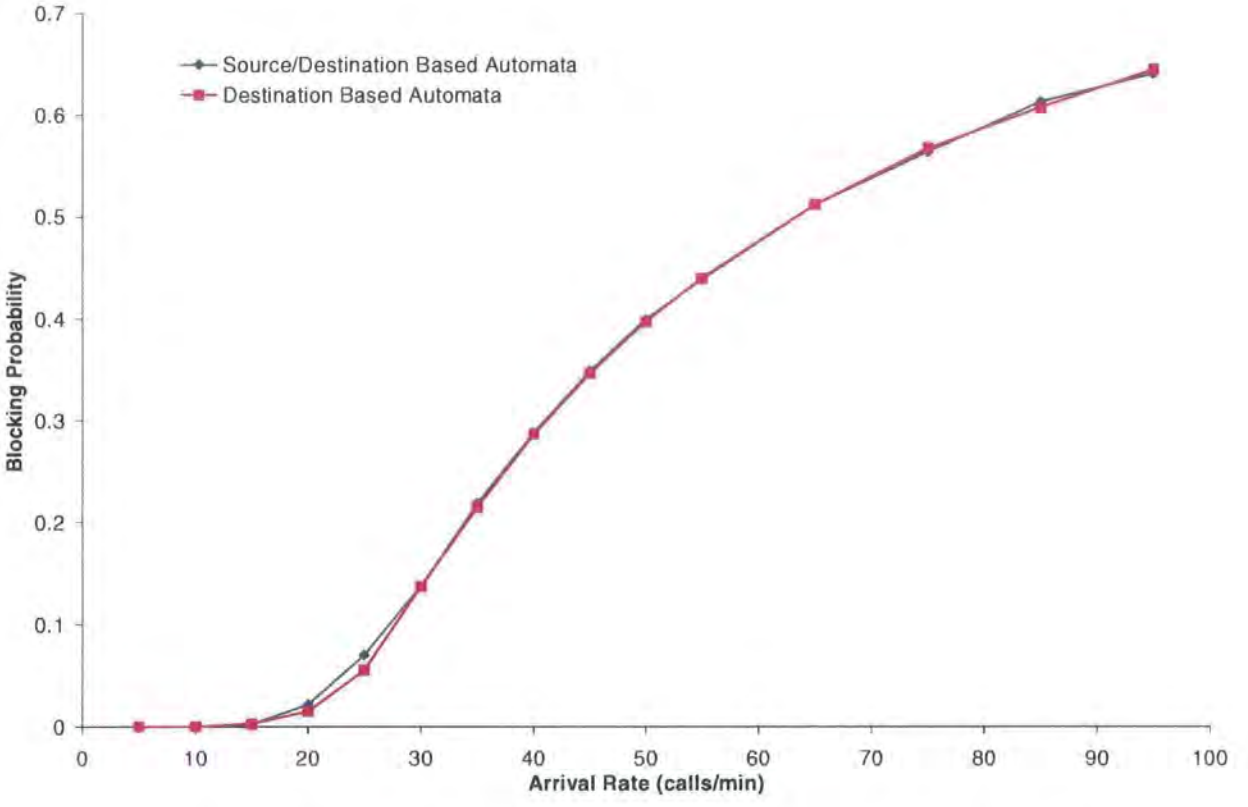


Figure 3.14 – Blocking Probability for different granularity automata.

3.7.6. Large Bandwidth Requests

For this experiment, we kept the average traffic loading the same (25 units/min), while increasing the bandwidth required by calls. Hence, the product of bandwidth required and call arrival rate was kept

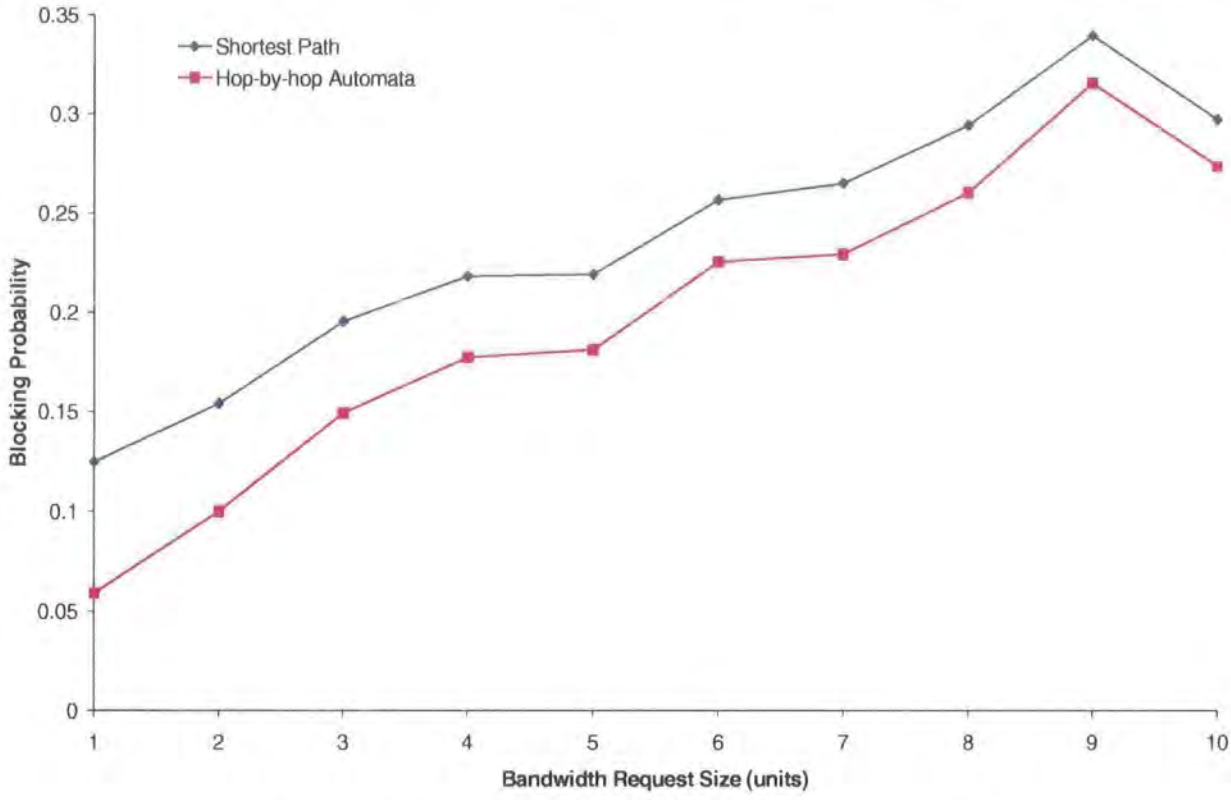


Figure 3.15 - Blocking probability, bandwidth request variation, 10 node network.

constant. For the 10 node network under even traffic demands, we show the blocking probability as a function of the size of bandwidth requests in Figure 3.15, for both shortest path and learning automata based routing with LRI reinforcement and learning rate of 0.03. For both curves, the blocking probability increases with bandwidth request size, since although we are keeping the overall load constant (i.e. bits/sec), the variance in the offered load is increasing, a result predicted by Erlang's formula (i.e. from a blocking perspective, it is better to have many small bandwidth connections than a few large bandwidth ones). One important point is that the performance difference between shortest path and automata routing decreases with increasing bandwidth request size, since shortest path routing reduces the bandwidth fragmentation, that is, instead of having small pockets of capacity at a large number of nodes, we have larger pockets of capacity at fewer nodes. This result has also been observed in [98]. Thus, the attractive properties of some form of alternate routing decrease at high network loadings, particularly if the traffic demands consist of only a small number of high bandwidth flows. As suggested in [98], it may be better to choose between a small number of pre-computed paths to reduce the degree of bandwidth fragmentation.

3.7.7. Multi-Rate Traffic

For multi-rate traffic offered to a single pipe of certain capacity, Erlang's formula can be augmented to calculate the blocking probability for each traffic class/rate (e.g. see [124] or [125]), although the

computational complexity increases exponentially with the number of traffic classes even for the single link case. As far as routing is concerned, should traffic types requiring different bandwidths be routed differently? In general, the optimal policy will be to reserve some resource exclusively for each traffic class whilst sharing the remainder (see [122]). If the traffic demands are very different in terms of arrival rates and bandwidth required, it may be better to bias towards isolation of resources rather than sharing. For scaling purposes however, it is better that one routing algorithm route all requests from all different traffic classes. Thus, although a small performance improvement (i.e. overall blocking probability) may be possible by using different learning automata for different traffic requests in order to provide better isolation, this will have a significant drawback in that it will not scale to a large number of traffic types and will increase convergence times for all the different automata. It should be remembered that the number of potential traffic classes is in no way bounded by the size of the network. In general, it should be the job of adaptive routing to react to localised resource shortages and provide alternative routing options in such cases. To achieve a desired level of blocking to each traffic type, it is the job of resource partitioning to allocate the appropriate level of resource to the range of traffic classes. In Figure 3.16, we show blocking probability for the case when the 10 node network was presented with two discrete traffic types. One traffic class had connections requiring 1 unit of capacity whilst the other required 2 units. Two routing strategies were implemented, the first utilised traditional automata routing where all requests were routed by one set of automata, and the second used discrete learning automata for each traffic (i.e. bandwidth) class. The blocking probability is plotted as a function of total nodal arrival rate in Figure 3.16 below. Each traffic class had an identical arrival rate. i.e. total arrival rate/2. In addition to blocking probabilities for each traffic class, the 'bandwidth blocking rate' takes the bandwidth of rejected sessions into account and is defined as follows:

$$\text{bandwidth blocking rate} = \frac{\sum_{i \in B} \text{bandwidth(i)}}{\sum_{i \in S} \text{bandwidth(i)}}, \tag{3.2.}$$

where B is the set of all rejected sessions and S is the set of all attempted sessions. The bandwidth blocking rate for the shortest path, discrete and combined automata schemes is shown in Figure 3.17. Hop-by-hop automata were used with LRI reinforcement (of 0.03) and traffic followed an even distribution. From both Figure 3.16 and Figure 3.17, we see that both learning automata mechanisms provide mild improvements over shortest path routing in the low-mid arrival rate range. The discrete and combined automata traces closely follow one another, and we suspect that one set of learning automata will be an adequate engineering solution for providing an adaptive routing capability in future networks. In Chapter 4, we address a similar problem examining the potential of using separate automata for the routing of both real-time and non-real-time traffics.

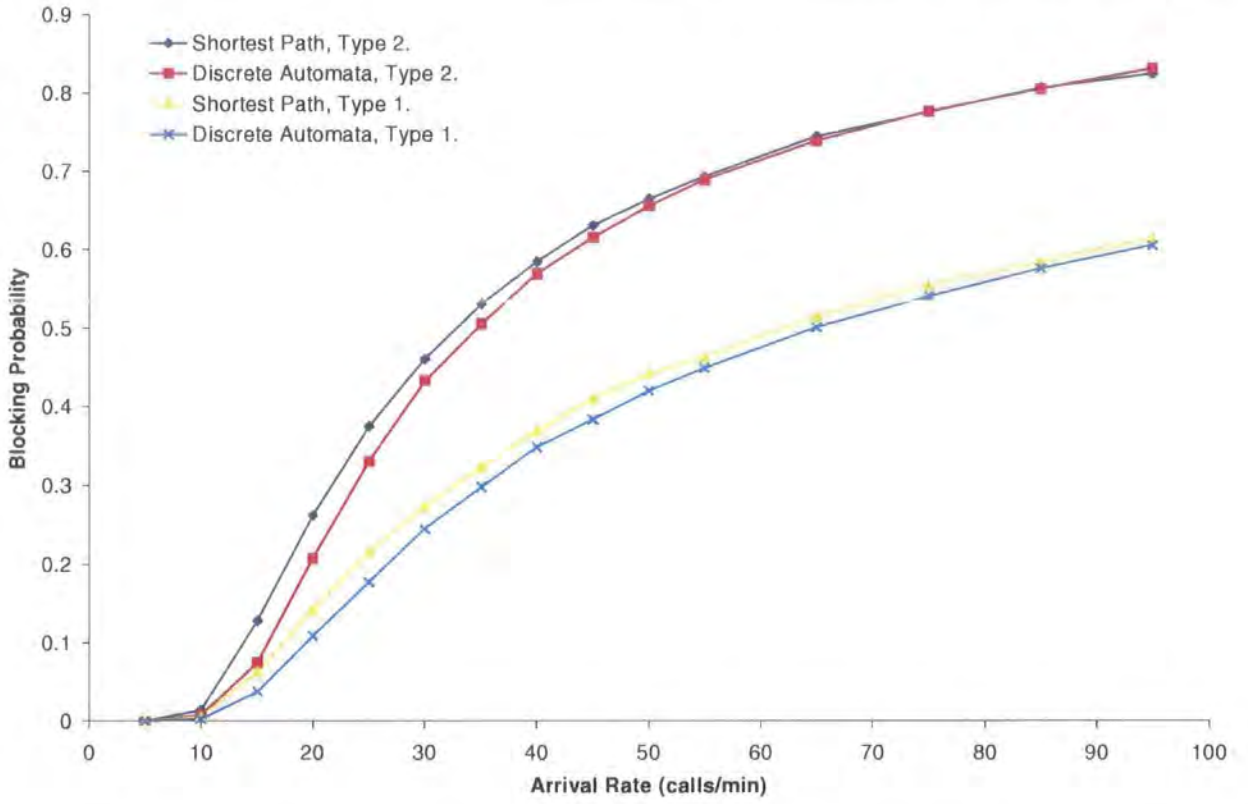


Figure 3.16 – Blocking Probabilities, 2 Traffic Types, Shortest Path and Discrete Automata routing.

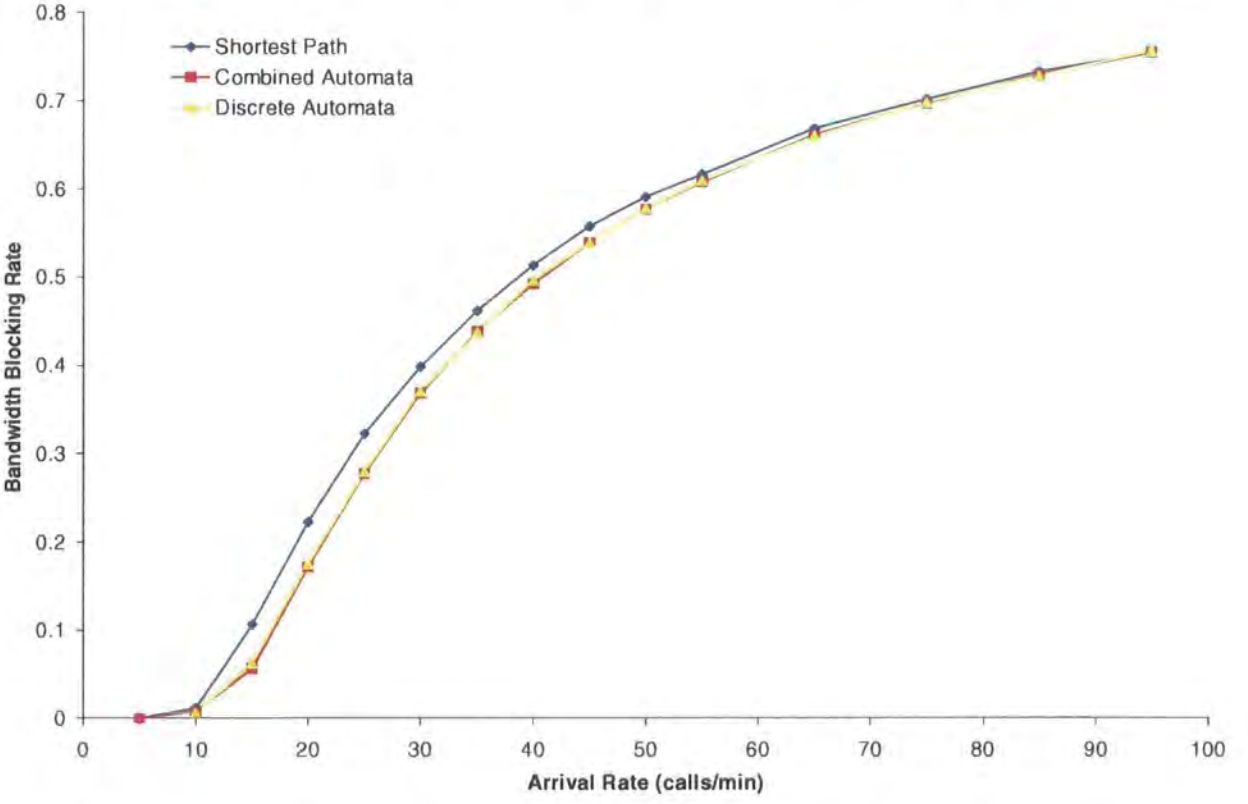


Figure 3.17 – Bandwidth Blocking Rate for 2 Traffic Types

3.7.8. Automata for aggregated/inter-domain routing

In the following experiment, we show the advantages of using learning automata based routing at the inter-domain level. The requirements for an inter-domain routing protocol tend to be quite different to those of an intra-domain protocol since individual domains may not wish to reveal internal topology details about their networks. Even if there are no 'policy' restrictions on passing information between domains, overall scalability of a routing architecture may not allow for very much information exchange. A routing 'policy' can be defined as a scheme run by the authority in charge of the domain, to restrict access of the network to a certain set of users. It is conceivable that policies could change fairly frequently in future integrated services environments as service providers compete for user custom. When a domain changes its policy, this policy change must also be distributed to other domains to maintain successful call set-ups. With a distributed learning approach however, we automatically adapt to changes in policy of other domains through call set-up feedback information and do not therefore require updates. To implement shortest path routing in this experiment, each domain nominated a node to send and receive traffic for each locally connected domain. These are sometimes known as 'border nodes' (e.g. similar to BGP [126]). In this way, we hide internal topology details of a domain from other domains. To implement a distributed learning automata scheme, each node now contained an automaton for each destination node in its own domain, and one for each neighbouring domain. For both automata and shortest path schemes, the ratio of information stored in the aggregated to unaggregated case can be written as :

$$\frac{N/d + d - 2}{N - 1}, \tag{3.3.}$$

where N is the number of nodes in the entire network and d is the number of (equal size) domains. For the aggregation experiments performed here, the 30 node network in Figure 3.3, was divided into 3 domains of 10 nodes each (see Appendix D). Thus, the information storage ratio was 11/29, or alternatively, an information storage reduction of approximately 2/3. In addition to reducing the storage requirement, the aggregation process will reduce convergence times since there are fewer overall automata in the network although the steady state (blocking) performance will be reduced due to the information loss. In Figure 3.18, we show the resulting blocking probabilities for aggregated automata and shortest path routing schemes. A purely random routing algorithm, assumed to be employed by all nodes in all domains is also shown for comparison purposes. For the automata, LRI reinforcement was adopted with a learning rate of 0.03.

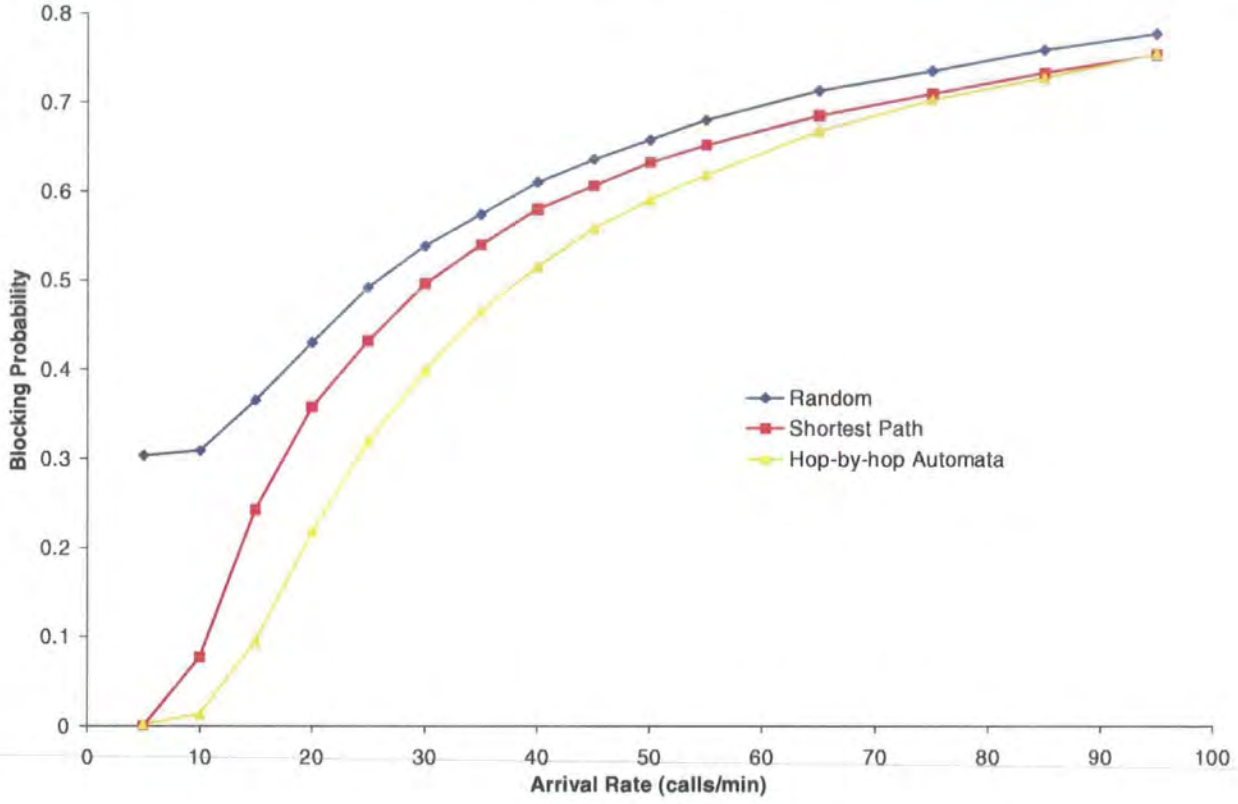


Figure 3.18 - Blocking probability, 3 domains, 30 node network.

Despite the level of aggregation and resulting information loss, distributed hop-by-hop automata based routing provides significantly lower blocking probability over a wide range of arrival rates through sensible use of alternate paths. Automata achieve these savings since they are able to learn of paths to other domains in addition to those that pass through the border nodes. In fact, in terms of percentage improvement, automata provide a greater improvement over shortest path routing for this example than for the intra-domain case studied earlier, mainly because the shortest paths are constrained to pass through the border nodes which quickly become congested as a result. We should note the closeness of the aggregated traces to the purely random strategy at mid to high arrival rates. This indicates that the aggregation process has limited our ability to improve upon a random approach other than at low loads. In general, aggregation will result in less information storage at the cost of the ability to adapt to the changing network state, generally leading to reduced throughputs in the steady state. Since we are likely to require considerable adaptability due to the high degree of uncertainty associated with future integrated services networks, this may place an upper limit on the amount of aggregation we might wish to adopt. Work within the IETF is considering scalability issues for routing in integrated services networks, and an architecture has been proposed known as the, 'Nimrod Routing Architecture', which has the ability to represent and manipulate routing related information at multiple levels of abstraction [127]. In general, the aggregation process introduces problems with regard to the accuracy of state information [93]. This further questions the use of dynamic state based QoS-routing schemes since the hierarchical state

gathered may not necessarily give an accurate representation of the available resources on a particular path.

3.7.9. Automata as a parallel/background optimisation process

One extremely attractive feature of automata based routing algorithms is that they may be used in parallel with other routing algorithms (e.g. shortest path). Since we are likely to have some form of feedback telling users the success/failure of connection requests, automata can process this feedback information in the background, the automata probabilities converging to a good route set. Such a scheme can also be useful for initialising the automata probabilities from a random state, helping to reduce the chance of routing loops occurring as the automata converge. We demonstrate this application of learning automata using hop-by-hop automata in the background to a shortest path algorithm. After 100,000 connection requests, we send a signal to the set of nodes informing them to switch to automata as the foreground routing algorithm. Upon switching, we freeze the learning rates (to zero) so we can observe how well the learning automata have learnt the network state whilst running in the background. We have simulated two alternative schemes for running automata in the background to shortest path routing. The first scheme simply updates the learning automata based on the feedback from the shortest path routing choices. For a shortest path scheme, each node always selects one output link based on the shortest path so that automata will only learn of the viability of that single output link. In order for automata to learn properly, we require the range of routing options to be explored. For the second scheme, we send out a 'hypothetical call attempt' once in every 10 real call attempts. This hypothetical call is treated exactly the same as normal calls (i.e. undergoes admission control) but does not reserve bandwidth in the network. The hypothetical call attempt effectively tells us, 'if a real call were sent out along this route, would it be accepted or rejected?'. For the second scheme, we update the automata probabilities entirely based upon the feedback response from the hypothetical call attempts. To perform the experiments, we assumed the same traffic and topology set-up as for the uneven traffic experiments in section 3.7.2. The automata used an LReP reinforcement scheme with a reward of 0.03 and a penalty of 0.005 to prevent the probabilities latching on to the shortest path routes. In Figure 3.19, we show sample paths of the blocking probability (measured every 100 connections) for both background learning schemes. Also shown are the steady state blocking probabilities for shortest path routing and automata routing with LRI reinforcement. The nodal arrival rate was set at 22.2 calls/min. The traces show the average blocking probability resulting from the application of a moving average window of width 5,000 connection attempts. In Figure 3.20, we show the resulting entropy traces for the two background learning schemes, where the entropy is measured every 100 connection attempts. We see that the standard learning automata operating in the background have learnt that alternatives to the shortest paths are desirable in this case and upon switching, they lead to considerably lower blocking levels. The automata using 'hypothetical call attempts' have learnt to reduce the blocking further since they can explore all possible routes in the

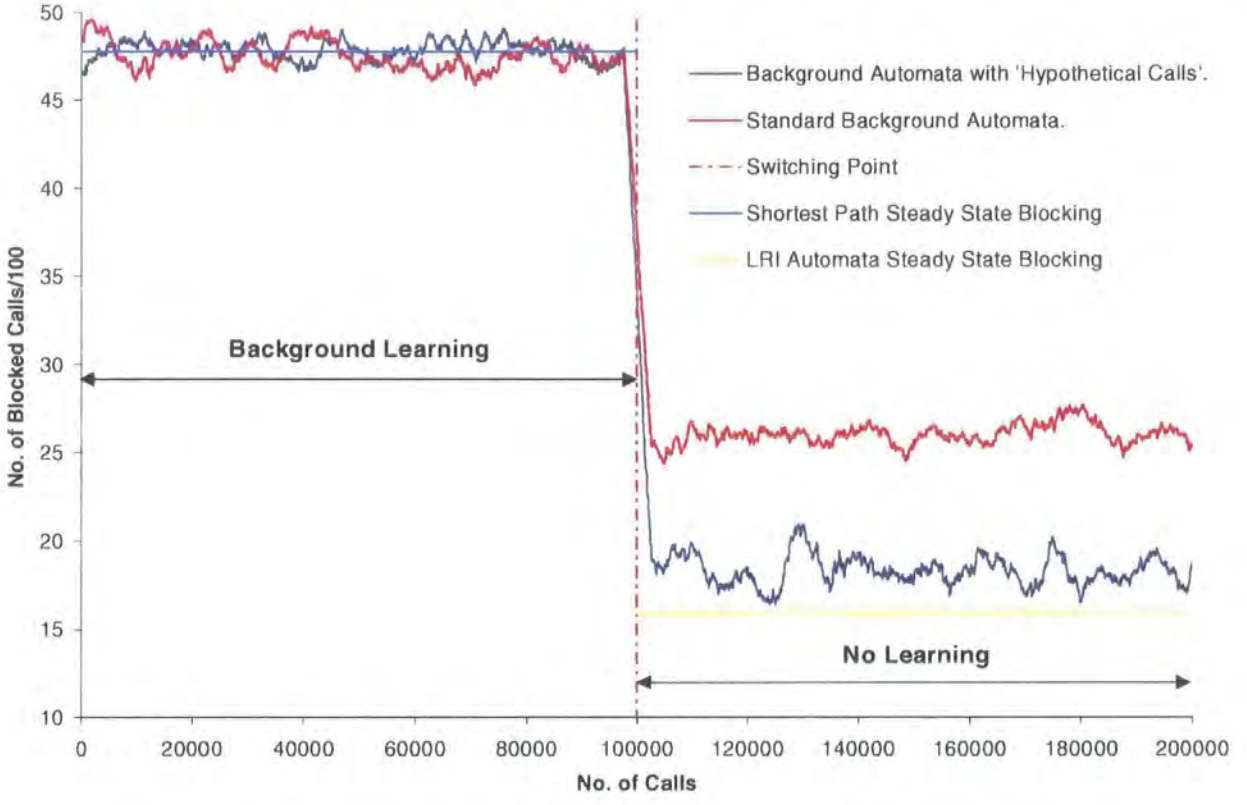


Figure 3.19 – Automata Background Learning Experiment, Blocking Levels.

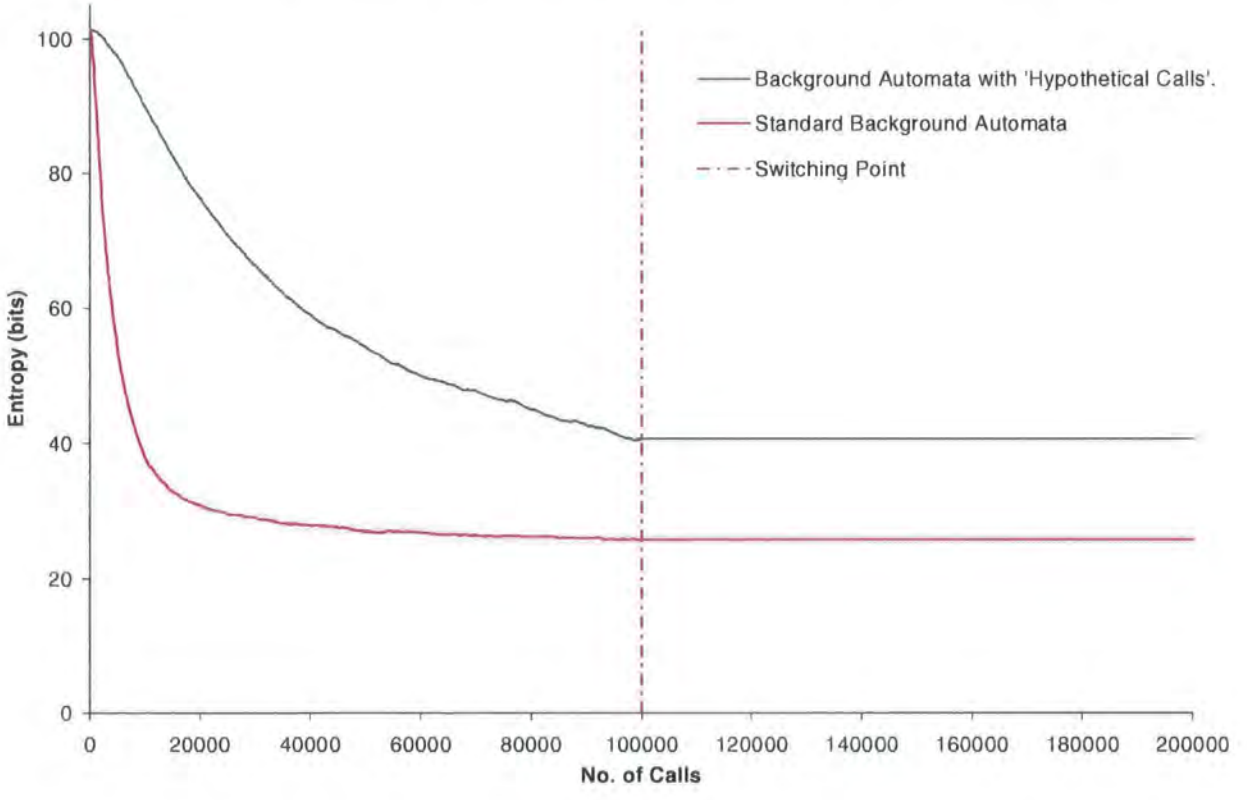


Figure 3.20 - Automata Background Learning Experiment, Entropy.

network whilst operating in the background, although they require that hypothetical call set-ups be sent out through the network, introducing a messaging overhead. From a convergence perspective, we see that the entropy in Figure 3.20 for the standard background automata scheme drops faster than the 'hypothetical call' scheme since we are updating the probabilities on every call attempt rather than effectively every call in 10.

Although we do not believe that there will be a centralised control node in future networks which gathers and processes state information for the entire network and installs the appropriate routes, there may well be a node responsible for instructing nodes which particular routing (or control) algorithm they should be using at any instant in time. Thus, if we switch over to an adaptive routing algorithm such as learning automata as in the above example, this node will have the ability to inform all nodes in the network to switch back to simple shortest path routing if so required.

3.7.10. Changing Resource Reservation Models/High Bandwidth-Delay products

In this section, we compare the different resource reservation models described in section 3.5.3. We compare the steady state blocking probability of shortest path and hop-by-hop automata routing when the propagation delays are 0.001s and 100s respectively. The traffic and topology set-up are the same as for the even traffic experiments in section 3.7.1. for the 10 node network. For the link propagation delay of 0.001s, the traditional set-up mechanism curves have been omitted since they are practically the same as for the 'on-the-fly' 1 curves, since the link propagation delay is very small compared to the mean holding time of connections (30 secs). Blocking probabilities for this case are plotted in Figure 3.21. For the case of link propagation delays of 100s, the two 'on-the-fly' signalling mechanisms are effectively the same from a blocking perspective since the connect 'reject' signal will not even traverse a single hop before the 'end of connection' packet is received. Thus, only the traditional set-up is plotted together with the 'on-the-fly' 1. Blocking probabilities for this case are plotted in Figure 3.22. From Figure 3.21, we see that the automata still provide similar improvements in terms of blocking when considering different resource reservation mechanisms. The 'on-the-fly' 1 mechanism produces lower blocking than the 'on-the-fly' 2 mechanism, since the prior technique removes the reservation state from previous nodes in the chain when a connection attempt is rejected, thus leaving more resources for future set-up requests. In Figure 3.22, we see the drawback of a traditional set-up mechanism when the mean holding time of connections is much less than the propagation delay. The blocking level is extremely high as even if we wish to send a small burst of data at a guaranteed data rate, we have to traverse the network twice before we can send any data. Previous studies of learning automata routing have often made the assumption of instantaneous feedback to the automata. In an environment with large bandwidth-delay products, an automaton may make a number of decisions before receiving any feedback (i.e. delayed feedback). In these cases, it may be necessary to reduce the learning rates to avoid any oscillatory behaviour, or the latching of the automata probabilities to 0 and 1. We have run simulations for the 10 node network under even traffic demands for different propagation delays and automata learning rates. The automata use LRI

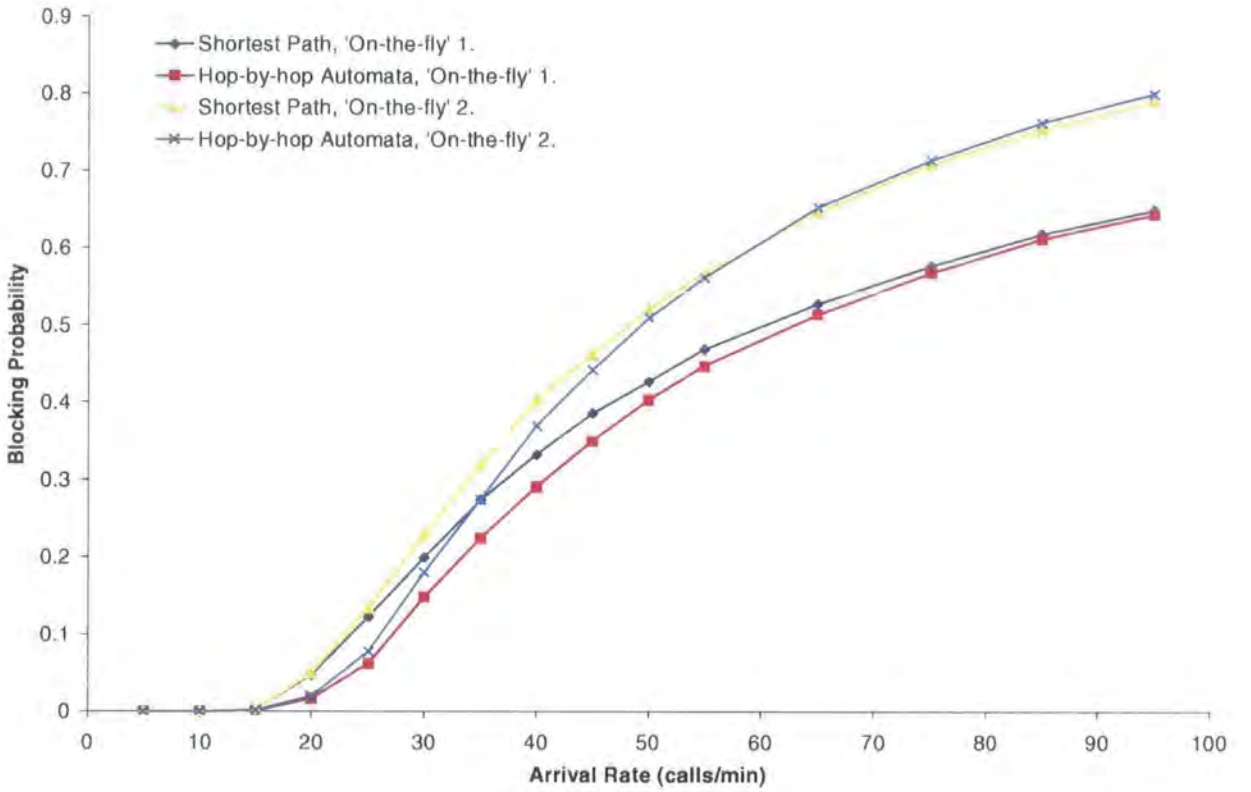


Figure 3.21 – Blocking Probabilities, different resource reservation models, prop. delay = 0.001s.

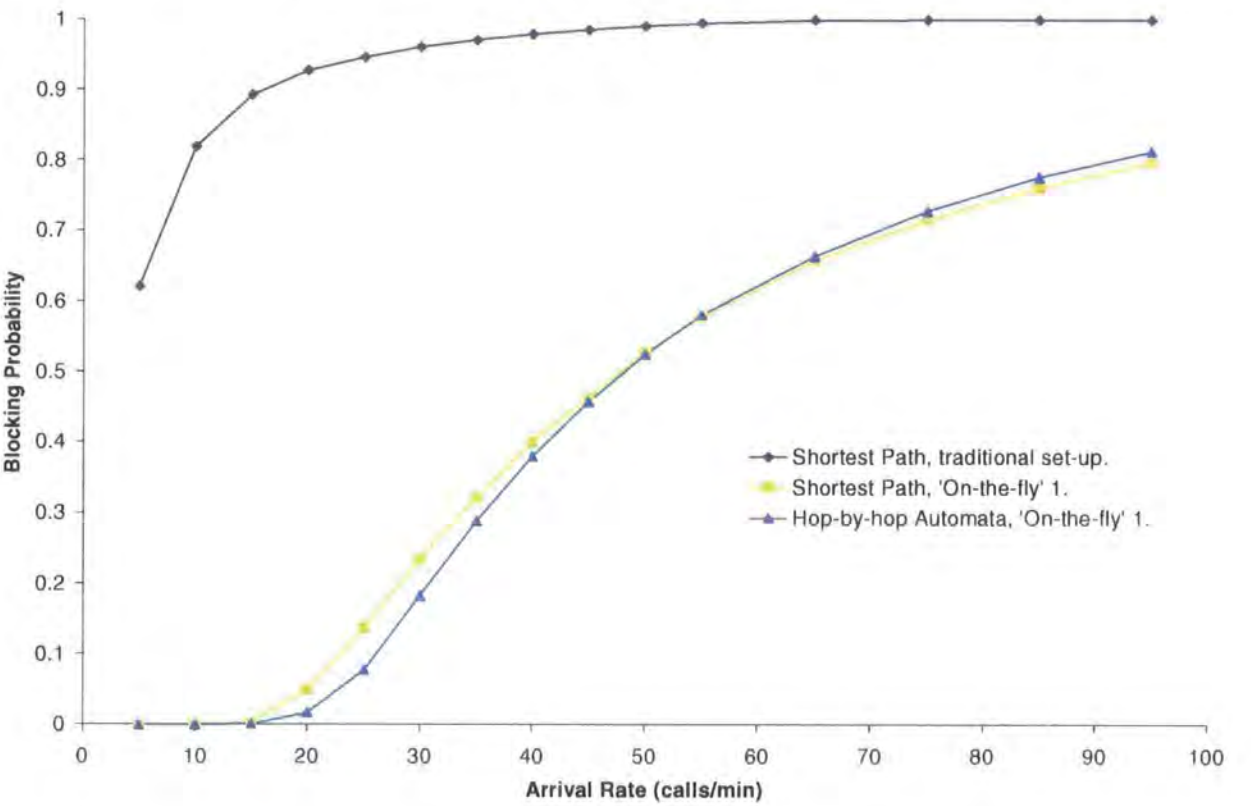


Figure 3.22 - Blocking Probabilities, different resource reservation models, prop. delay = 100s.

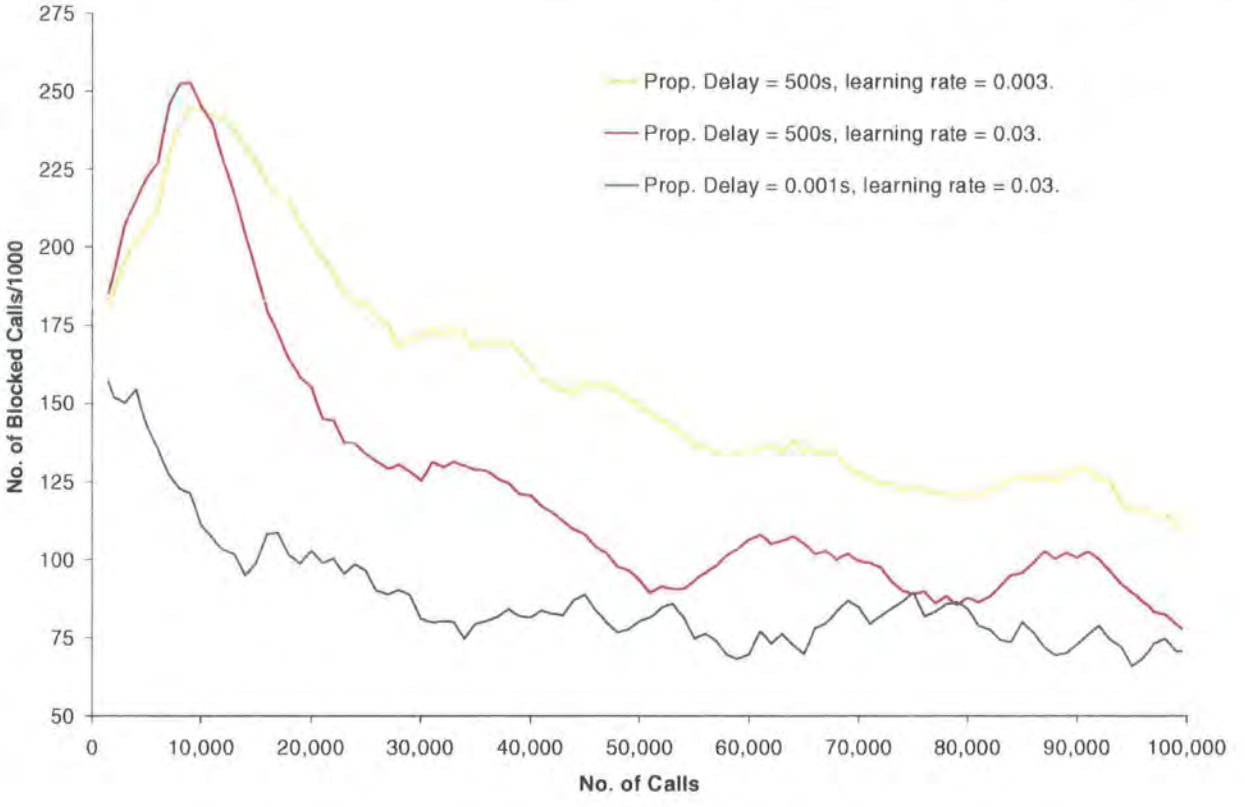


Figure 3.23 – Blocked Calls, changing prop. delay and learning rates.

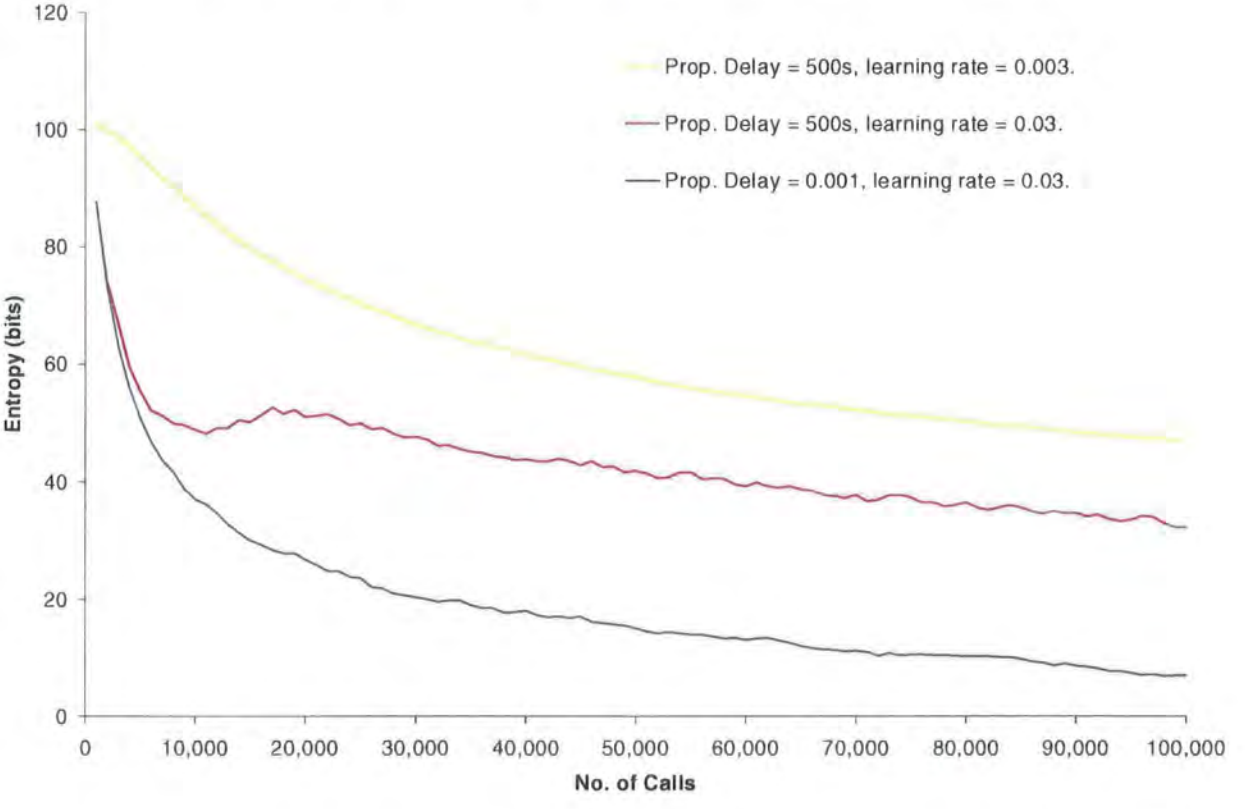


Figure 3.24 - Entropy, changing prop. delay and learning rates.

reinforcement and the 'on-the-fly' 2 resource reservation mechanism. For link propagation delays of 0.001s and 500s, we have recorded sample paths of the number of blocked requests (per 1000), entropy and cumulative routing loops for learning rates of 0.03 and 0.003. The nodal arrival rate is set at 25 calls/min. In Figure 3.23, Figure 3.24 and Figure 3.25, we plot the blocked requests, entropy and cumulative routing loops respectively.

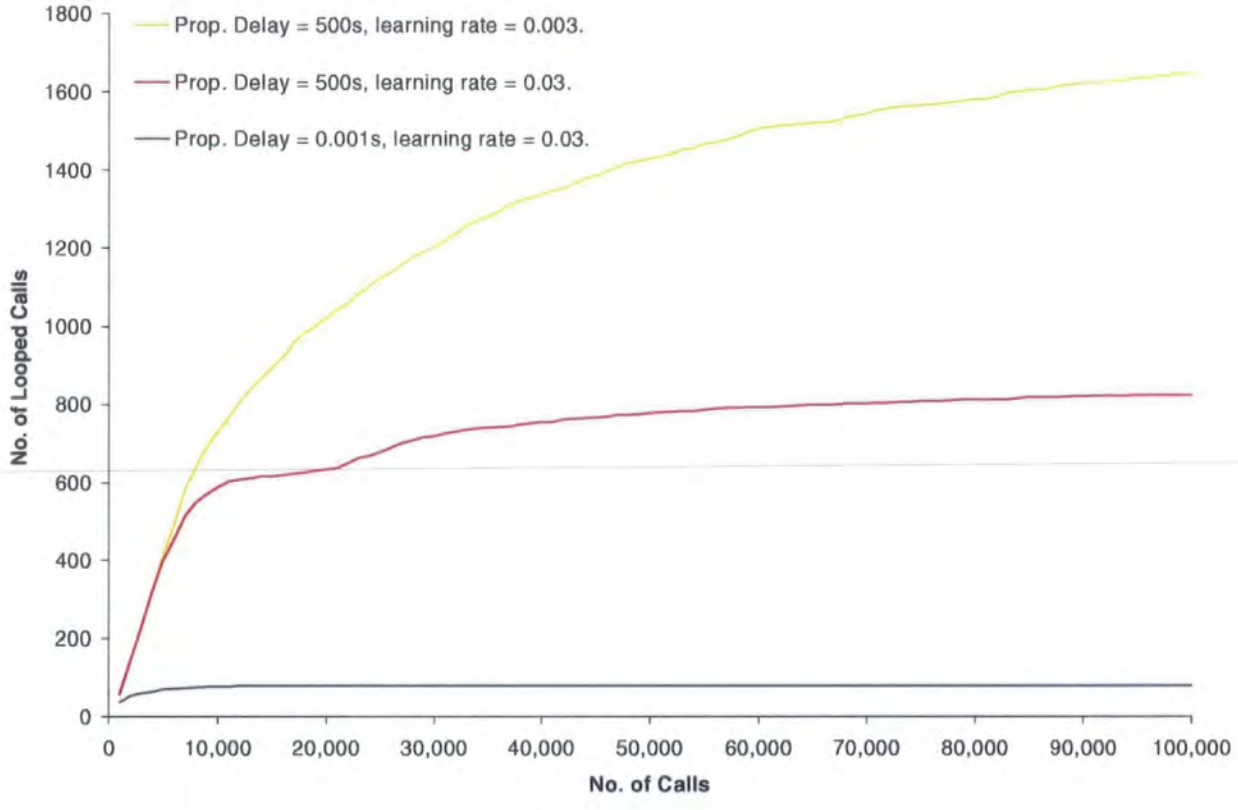


Figure 3.25 – Looped Calls, changing prop. delay and learning rates.

We see that the effect of increasing the propagation delay is to effectively extend the convergence period as the behaviour of the automata probabilities becomes more oscillatory due to the delayed feedback, observed by the extended entropy traces and increased number of routing loops in Figure 3.24 and Figure 3.25 respectively. We can see the oscillation from the entropy trace in Figure 3.24 for the automata with a learning rate of 0.03 and propagation delays of 500s. The trace can be seen to dip, then rise again as the automata probabilities move from their original choices to new choices. If the propagation delays are large enough, the probabilities of the LRI based automata will simply become locked to 1, effectively negating the point of any learning capability. To combat this, the learning rates must be reduced although this will extend the number of iterations required to converge as observed in the three Figures.

3.8. Summary

In this chapter, we have reviewed the problem of routing of real-time connections (QoS-based routing) in Integrated Service networks, where 'real-time' traffic is assumed to require a resource reservation process. We have shown the typical savings achieved by hop-by-hop and source routing learning automata through their sensible use of alternate paths for two realistic network topologies, in terms of blocking probabilities, and have considered the complexities of their implementation. In addition, we have examined the learning automata performance under more complex assumptions regarding future networks including multi-rate traffic, aggregated routing, trunk reservation issues and routing granularity. It is likely that learning algorithms will be most appropriate as a complement to existing algorithms and we have shown how learning automata have the ability to operate 'in the background', learning good route sets from the simple binary feedback based upon whether calls are accepted or rejected. Finally, we showed that there may be issues when considering the operation of learning automata in high bandwidth-delay product networks, and that the learning rates must be set carefully to avoid the latching or oscillation of the automata probabilities.

In the next chapter, we consider how automata may be used for the routing of non-real-time or best-effort traffic. For a simple network architecture containing real-time and non-real-time traffics, our aim is to investigate the benefits of some form of adaptive routing for both traffic types. Specifically, we are interested in whether separate routing algorithms should be used for each traffic class or whether the benefits of adaptive routing can be accrued by using one routing algorithm for both traffic types.

Chapter 4

Learning Automata for Routing of NRT and mixed traffics

4.1. Introduction

In this chapter, we provide a brief review of how automata may be used to route NRT or best-effort (packet switched) traffic. We present results showing how automata learn to minimise packet delays in a 10 node network subject to traditional and empirical traffic models. Additionally, we consider a mixed traffic environment containing both resource reservation based (or real-time (RT)) and best-effort (or non-real-time (NRT)) traffics, where the resource reservation traffic has priority over the best-effort traffic. Here, we investigate the combination of routing algorithms which minimise the delay of the best-effort traffic under various traffic mixes and distributions.

4.2. Learning Automata for NRT routing

Optimal routing can be defined as the problem of choosing paths for traffic flows to minimise overall packet delays in the network. Given that we have knowledge of the external arrival rates of traffic and the link capacities, a cost can be assigned to each link based on the expected delay for a M/M/1 queue, and the cost can be minimised using some iterative computation [90]. The traffic demands are rarely known with any accuracy however and an adaptive routing capability which uses some measure of the dynamic network state (e.g. queue lengths) can improve performance by adapting to the statistical fluctuations in the traffic demands (see Chapter 2). Adaptive routing has a long history in data networks. Routing decisions on the original Arpanet adapted to changes in network load, whereby link metrics were a function of the instantaneous queue size. Nodes exchanged routing tables with their neighbours every 600 milliseconds and computed minimum cost paths to the destinations. Due to the large variance in queue length samples, this metric was found to be a bad indicator of expected delay, and rapidly changing link costs led to the formation of routing loops [128, 129]. This distance vector algorithm was then replaced by a link state algorithm where the metrics used were now direct measurements of delay over 10 second intervals. Problems with route oscillation persisted however, and further modifications were proposed to make the computations of link metrics less responsive to the measured delays [128]. Oscillations in shortest path delay based algorithms were also shown analytically in [130]. The problems

with oscillations for adaptive routing on data networks led to the adoption of the current shortest path algorithms (e.g. RIP [92], OSPF [68]), which only adapt to changes in topology and not loading. The main problems with shortest path based adaptive datagram routing are that each packet may follow a different path, such that we can switch between alternate shortest paths almost instantaneously. The problem is exacerbated in that we can only use a single shortest path at any one time and not split the load amongst multiple paths. Although there have been problems with oscillations, some form of lightweight adaptive routing could still be an attractive proposition, since on a mixed traffic network containing RT and NRT traffics, adaptive control of a NRT element can absorb the statistical fluctuations in the NRT and the RT elements. We feel however that an adaptive routing algorithm should split the traffic over multiple paths, and will operate over reasonably large timescales to avoid problems with potential oscillations as well as avoiding interaction with network flow/congestion control mechanisms.

Learning Automata have been applied to both the routing of virtual circuits and datagrams. Automata for routing of virtual circuits are examined in [83], [84], [85]. Both hop-by-hop and source routing automata have been used to route the virtual circuit set-up packets, where subsequent data packets from the session follow the installed path. In these implementations, the automata use feedback regarding the average delay of the virtual circuits to update their action probabilities. S-type automata are used since delay is a continuous variable and the response from the environment can take any value in the region (0,1) (see Appendix A). Since flow/congestion control protocols often operate end-to-end (e.g. TCP [131]), it is possible to 'piggyback' delay feedback information onto the congestion control feedback signals. Studies on reasonably complex networks have shown that automata minimise queuing delays through load splitting [85]. Like the source routing automata examined for routing of real-time traffic, source routing virtual circuit automata will generally require the complete path to be specified at circuit set-up, which means that nodes must maintain global topological information. However, if we have the capability of source routing of real-time traffic, the source routes can be used by the NRT routing algorithm at very little incremental (storage) cost.

For datagram networks (e.g. IP networks), each datagram is routed as a separate entity and routing decisions are carried out by independent automata situated at each node. Automata for routing of datagrams has been examined in ([85], [86], [87], [88], [89]). Routing on reasonably large scale networks [85] has shown that the automata minimise queuing delays through load splitting as with the virtual circuit studies. Stability problems are not usually encountered since the learning rates are set quite low so that it normally takes some thousands of packet transmissions to converge [85]. Here, we describe the approach adopted in [85].

In order to realise a global feedback signal, each node is required to maintain delay estimate vectors, which contain the average delay between the current node k and each destination node j. Since a different route to j may exist for each routing option (output link) at k, a separate delay vector is maintained for each outgoing link or routing option, giving $\hat{d}_j^{ka}$ as the estimated delay between nodes k and j, assuming

option (or link) a is chosen by node k. For the feedback mechanism to work correctly, the automata routing scheme relies on a small control packet to be sent to the previous node by a node receiving a data packet as shown in Figure 4.1. This control packet contains the delay feedback, which is used to update the automata action probabilities and the delay estimates at the previous node. The delay value passed by the control packet from node n to node m in Figure 4.1 consists of two components. These are a local delay measured between the two nodes, and the delay estimate $\hat{d}_j^{na}$, from node n to the destination node j given that the data packet was routed on output link a. In this way, the global delay diffuses through the

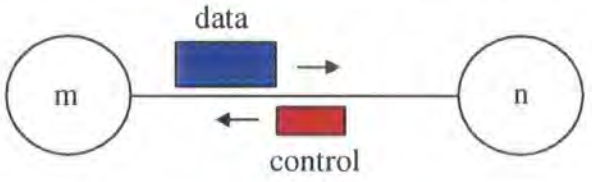


Figure 4.1 - Data and acknowledgement packets.

network as packets are transmitted between nodes, the learning automata action probabilities converging to a suitable routing strategy. Each node m stores an automaton for each destination node j, giving A_j^m. In Figure 4.2, the automaton at node m for destination j, A_j^m, selects an output link according to the automaton probability vector. Here, the automaton selects node n, this being reached after a local delay d_{mn} (queuing plus transmission plus propagation delay). A further routing decision is performed by the automaton at node n (for destination j), A_j^n. This automaton selects link a.

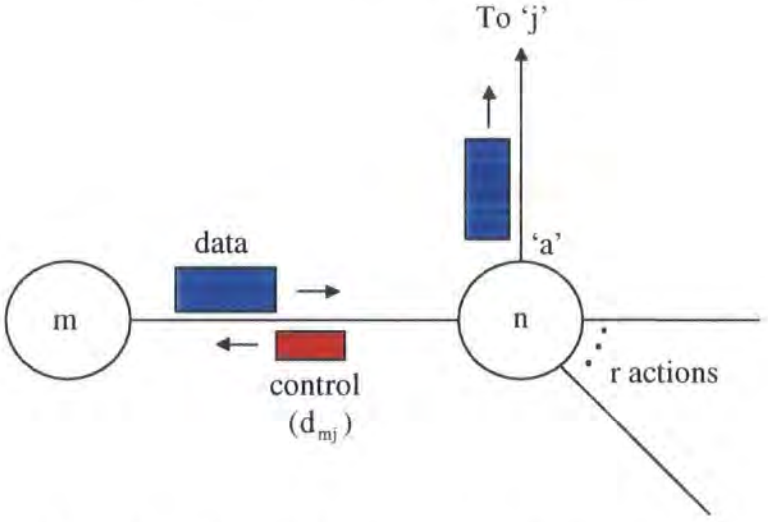


Figure 4.2 - Learning Automata for datagram routing

A collection of the path delay estimate can now be formed and sent back to node m via a small control packet. The global delay used for updating at node m, d_{mj}, is then calculated as follows.

$$d_{mj} = d_{mn} + \hat{d}_j^{na} \qquad (5.1.)$$

The actual response b_j^m to the automaton at node *m* (for destination *j*) is the normalised delay d'_{mj}.

$$b_j^m = d'_{mj} \tag{5.2.}$$

As well as modifying the action probabilities of A_j^m, the feedback delay is used to update the delay estimates at node *m* given link *a* was chosen, $\hat{d}_j^{ma}$. The new estimate of $\hat{d}_j^{ma}$ can be formed using the widely used exponential averaging technique as follows.

$$\hat{d}_j^{ma}(\text{new}) = \varepsilon \hat{d}_j^{ma}(\text{old}) + (1-\varepsilon) d_{mj} \quad 0 < \varepsilon < 1 \tag{5.3.}$$

For the experiments reported here, ε had a value of 0.99. Finally, with the automata used in the S model, the delay feedback must be normalised into the range (0,1). One way of achieving this is to divide by a normalising factor d_{max} so that the normalised delay d'_{mj} becomes :

$$d'_{mj} = \frac{d_{mj}}{d_{max}} \tag{5.4.}$$

This scheme requires prior knowledge of the maximum delay or normalising factor d_{max} however. One method which doesn't require *a priori* information is given by the function :

$$d'_{mj} = 1 - \frac{1}{\sqrt[n]{d_{mj}/d_{min}}} \tag{5.5.}$$

This function uses the minimum recorded delay, d_{min}, to map the feedback delay into the region (0,1). Figure 4.3 shows this function for a number of values of n.

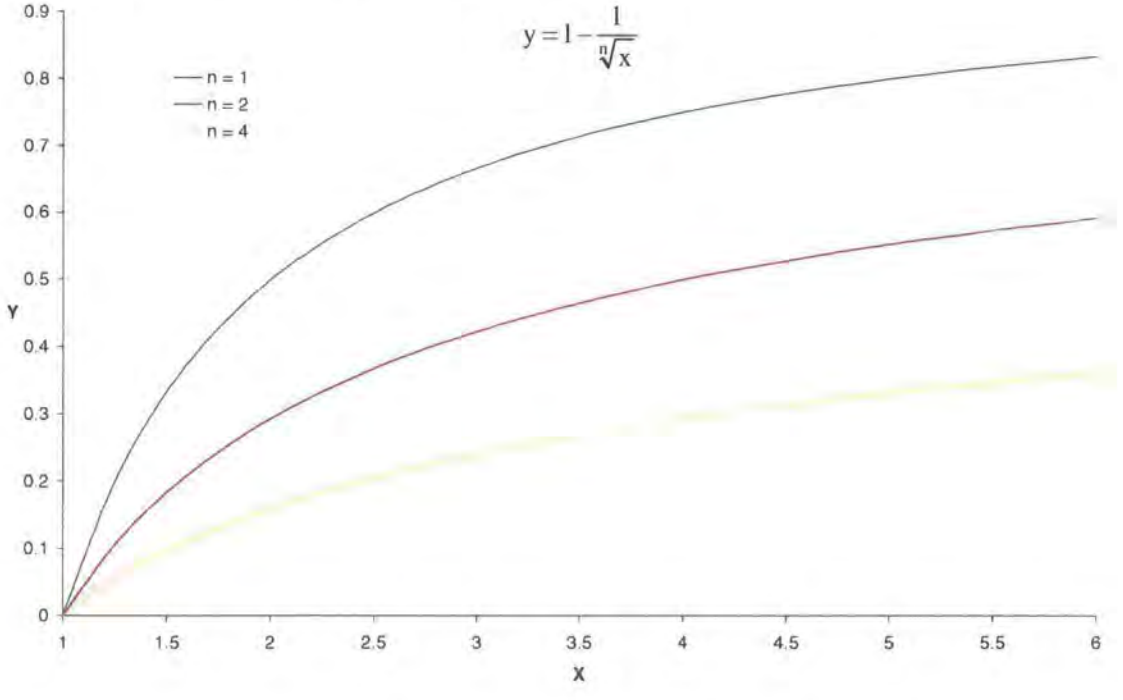


Figure 4.3 - $y = 1 - \frac{1}{\sqrt[n]{x}}$ for n=1, 2, 4.

This non-linear mapping of the environmental response has the effect of compressing the value of the response for large delay values but gives a large differential in response values for smaller values of delay so that automata take less interest in decisions leading to large delays but update the probabilities more vigorously for lower delays. For the experiments in this and later chapters, a value of n = 2 was adopted giving a square root law.

The technique described above relies on a small acknowledgement control packet to be sent from the current node to the previous node upon receiving a data packet. We believe that such a packet could be made extremely small to minimise bandwidth consumption, since the packet need not contain any routing overhead, only the node it has come from and the delay value described above. Although a protocol such as the Internet Protocol (IP) [120] does not explicitly cater for such a feedback mechanism, the method could be readily incorporated into an existing management protocol such as ICMP (Internet Control Message Protocol) [132]. In practice, it may not be feasible to send feedback for every packet sent, but rather, to send a feedback acknowledgement when a certain number have been sent, say 10 or 100 packets. In this way, we use less signalling bandwidth at a cost of slower convergence to the changing network state.

4.3. Simulation Set-up

Traffic and topology information for the simulation experiments were read in via configuration files at simulation start-up as described in the previous chapter. For the traffic, an even traffic distribution was initially adopted where all nodes sent to other nodes at the same rate. All data packets were 416 bits and control packets were 41 bits (~10 times smaller). This is a significant departure from previous studies since it has often been assumed that the size of acknowledgement packets is negligible or that control packets receive priority service over data packets at the servers. We do not make such assumptions since we feel that control and data signals will be treated no differently for a best-effort service as with the current Internet, so that the control packets in the queues will add to the delay of the data packets. For automata based datagram routing, packets may circle indefinitely following a sequence of bad automata decisions during the convergence period. We therefore discarded packets at a node if they travelled more than 6 hops (maximum shortest path length for 10 node network is 4 hops) to ensure proper convergence to a sensibly short path set. We also constrained the automata to choose an output link for the data packets different to the one that the packet came in on to prevent trivial routing loop formation. For the initial simulations, packet interarrival times were drawn from an exponential distribution (i.e. Poisson process) and the processing rate of all nodes was set to 25kbit/s. All queues had infinite buffers.

4.4. Automata Routing of NRT Traffic

In Figure 4.4, we compare the average received packet delay as a function of the arrival rates for learning automata, shortest path and random routing for even traffic demands on the 10 node network in Chapter

3, Figure 3.2. Random routing can be considered as automata based routing with the learning rates set to 0, so that an incoming packet has equal probability of being forwarded over each output link. Automata were initialised in a random state, i.e. equal probability across all actions. Automata using LRI and LRP reinforcement algorithms were both simulated, where the reward parameter for the LRI automata was set to 0.03 and the reward and penalty parameters for the LRP automata were both set to 0.01, the learning rates being determined after a tuning process.

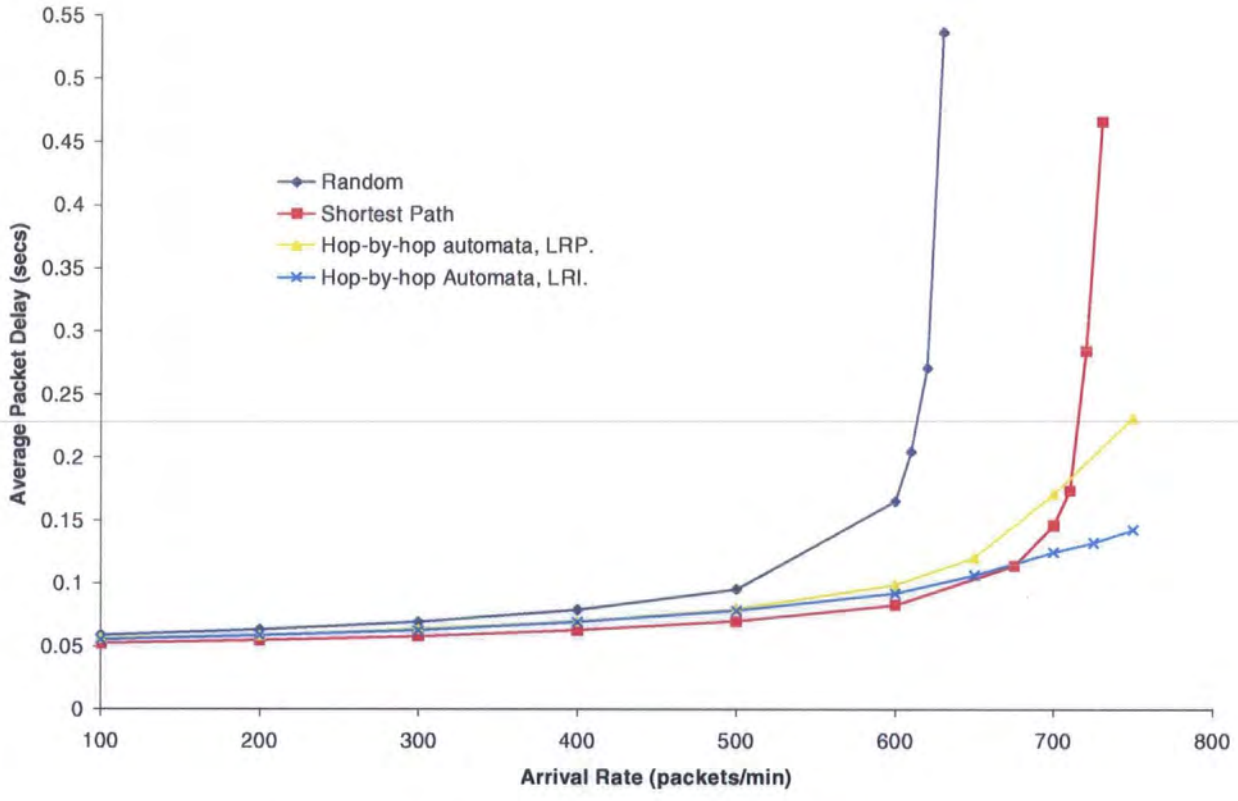
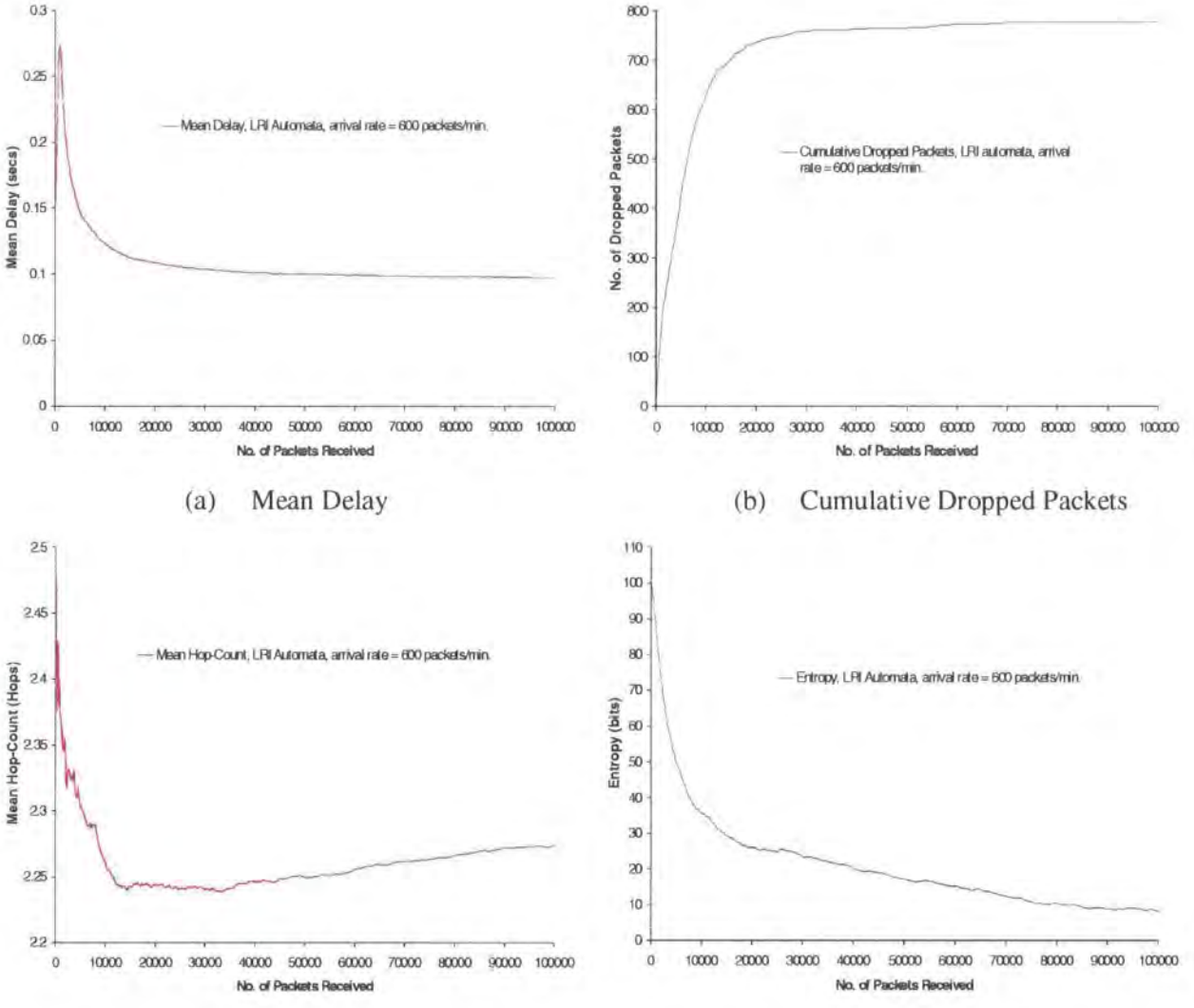


Figure 4.4 – Average Packet Delay, 10 Node Network.

At low loads, shortest path routing minimises average packet delay since the delay is largely due to transmission delays and using longer alternate paths would increase transmission delays. Previous simulation studies (see [85]) have suggested that automata are comparable with shortest path routing at low loads, although these studies have ignored the effect of the control packets which add delay to the data packets. At higher loads, queuing delay becomes the predominant component of end-to-end delay and the ability of learning automata to split the traffic produces lower average packet delays. The automata using LRP based reinforcement produce higher average packet delays than the LRI automata. Since LRI automata are ε-optimal in stationary environments whilst LRP automata are ergodic (Appendix A), we expect the LRI automata to converge to the static splits leading to minimum packet delays providing that the learning rates are low enough. Indeed, in Chapter 3, we observed how the LRI automata followed the optimal static probabilistic split for a simple two path routing problem. A number

of transient measurements were made to depict how the automata converge through self-organisation. In Figure 4.5, we show sample paths of packet delay, cumulative dropped packets, average path length and automata entropy.



(a) Mean Delay

(b) Cumulative Dropped Packets

(c) Mean Hop-Count

(d) Entropy

Figure 4.5 – Sample paths of delay, dropped packets, average path length and entropy.

It can be seen that the set of learning automata converge reasonably well after approximately 50,000 packets received for all 4 traces. Although prone to routing loops in the unconverged (random) state, routing loops disperse as the automata self-organise as shown through the number of cumulative dropped packets which approaches a constant.

In another test, we used the TCPLib traffic generation package [133]. TCPLib is an empirically derived traffic generation model based on studies of real TCP/IP network traffic flows for various network services such as TELNET, FTP etc... It has been found to accurately model the high variance associated with real network traffic [134]. For the tests performed, the TELNET function was used to generate one way packet interarrival times. In Figure 4.6, we show the average delay plots for random,

shortest path and automata based routing schemes using the TELNET traffic generator. For reference, we also include the shortest path delay for the Poisson traffic studied previously.

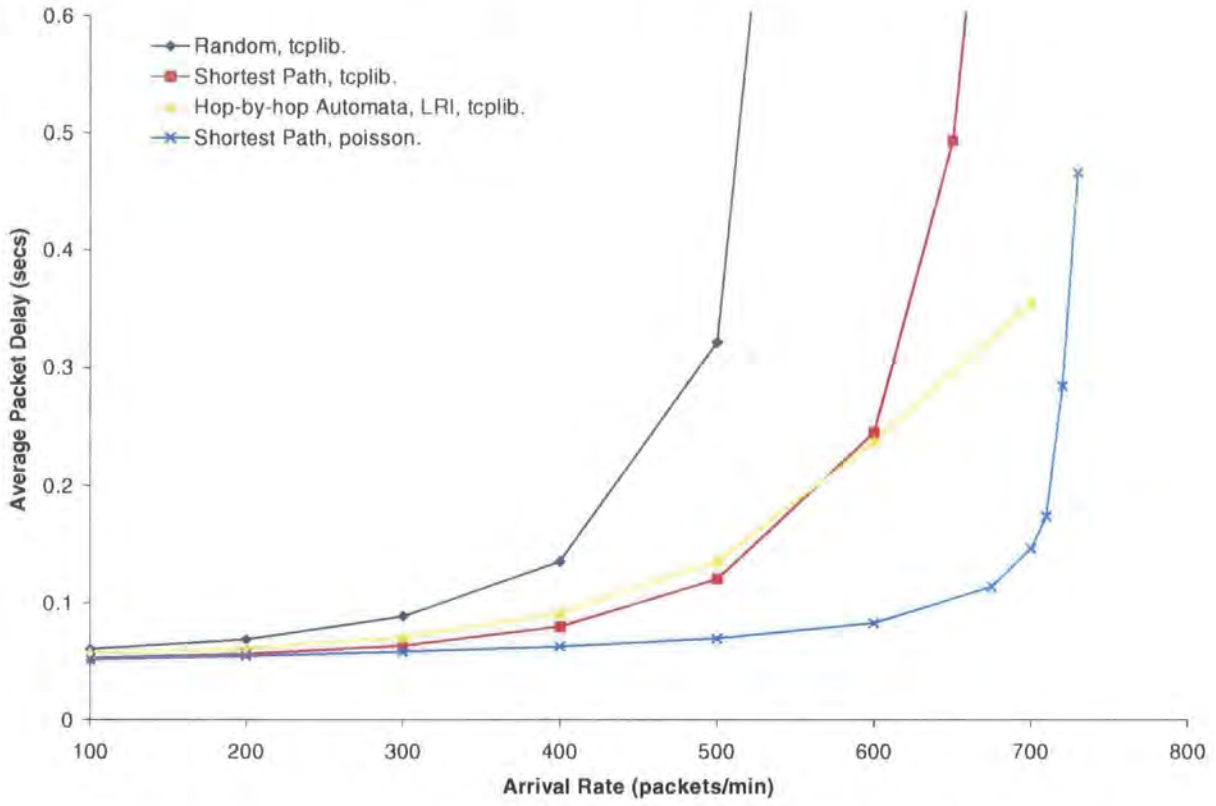


Figure 4.6 – Average Packet Delay, 10 Node Network, TCPLib TELNET Traffic.

It can be seen from Figure 4.6 that the resulting routing traces have the same relative behaviour as before, with learning automata giving superior delay performance at higher loads. Additionally, the absolute average delay of the TELNET generated traces are significantly greater than the previous Poisson case. This is due to the high variance associated with the TELNET traffic, which leads to longer average queue lengths and subsequent delays. These experiments have ignored the effect of congestion/flow control, which regulate the traffic flow of the end-stations dependent on the congestion within the network. If a flow control protocol such as TCP [131] is used over any adaptive routing protocol, there exists the possibility that the two control loops may operate on similar timescales, resulting in possible control loop interaction, leading to lower average throughputs. We believe that automata will be designed to operate on much longer timescales than end-to-end flow control protocols however, such that potential interactions are avoided. As bandwidth-delay products increase, the time to converge for flow control and adaptive routing become dependent on the propagation delays in the network. In this case, the learning rates for learning automata must be chosen carefully to avoid interaction. Another issue is that learning automata deliver a certain fraction of packets out of sequence whereas a single shortest path algorithm generally will not. This may need to be taken into account for the design of the flow control mechanism.

4.5. Automata Routing of Mixed Traffic

Here, the aim was to simulate a truly integrated service environment with two separate traffic classes. The first traffic class consisted of resource reservation based traffic (RT) as examined in the previous chapter. This traffic was assumed to have priority over the second traffic class which consisted of simple best-effort or NRT traffic as simulated previously. The resource reservation traffic was still modelled at the connection or call level to avoid excessively long simulation times and avoid having to simulate scheduling at the packet level. The best-effort traffic came into a single FIFO queue at the nodes, the service rate of which was regulated by the amount of bandwidth unused by the RT traffic. This model assumed that the bandwidth consumed by RT connections did not vary drastically for the duration of the call or that there were enough connections in progress to smooth out such fluctuations. i.e. the aggregate RT traffic has reasonably low variance. This modelling scheme is depicted in Figure 4.7 where the NRT traffic comes into a FIFO queue with a service rate of (B-W), where B is the capacity of the switching node and W is the aggregate bandwidth utilised by the RT traffic.

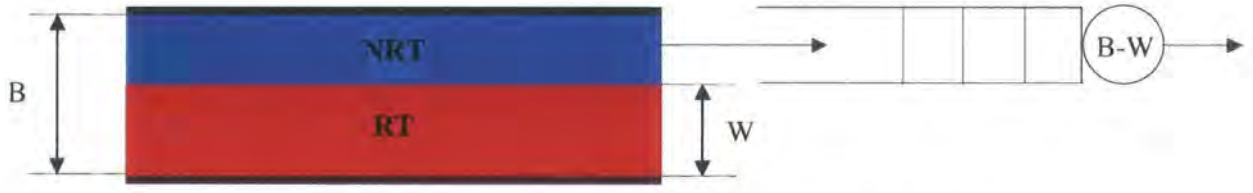


Figure 4.7 – Priority scheme for mixed traffic simulations.

The bandwidth available to the NRT traffic is now a function of the traffic statistics and routing algorithms of both types of traffic on the network. We have compared the performance of different combinations of routing algorithms in terms of the average packet delay for the best-effort traffic. Four of these schemes are the possible combinations of shortest path and learning automata based routing. The fifth scheme which we call 'RTmix', uses the RT automata probabilities to route the NRT traffic as well, where hop-by-hop automata were used to route the RT traffic. In this case, we only need to store one set of learning automata in the network rather than one per traffic class. One of the advantages then of using learning automata to route the RT traffic is that it provides a simple load splitting strategy which can be re-used for the NRT traffic via the automata probabilities. For the initial simulations, both RT and NRT traffics followed an even traffic distribution on the 10 node network. For the initial results, the nodal arrival rate for resource reservation (RT) traffic was maintained at 25 calls/min. All nodes had a capacity of 50 units where each unit corresponded to 5kbit/s bandwidth. RT traffic had potential access to 100% of the network bandwidth and the NRT traffic had access to the remainder. The five routing schemes are plotted in Figure 4.8. The naming convention used references the RT routing algorithm first and then the NRT. For example, sp/lri refers to using shortest path routing for RT and (LRI) learning automata for NRT. For all learning automata based routing, the LRI reinforcement algorithm was employed with a learning rate of 0.03. To measure the steady-state delay of the best-effort traffic, we first launched the RT

traffic and let the RT automata converge. We then launched the NRT traffic and measured the steady-state delay after another transient removal process.

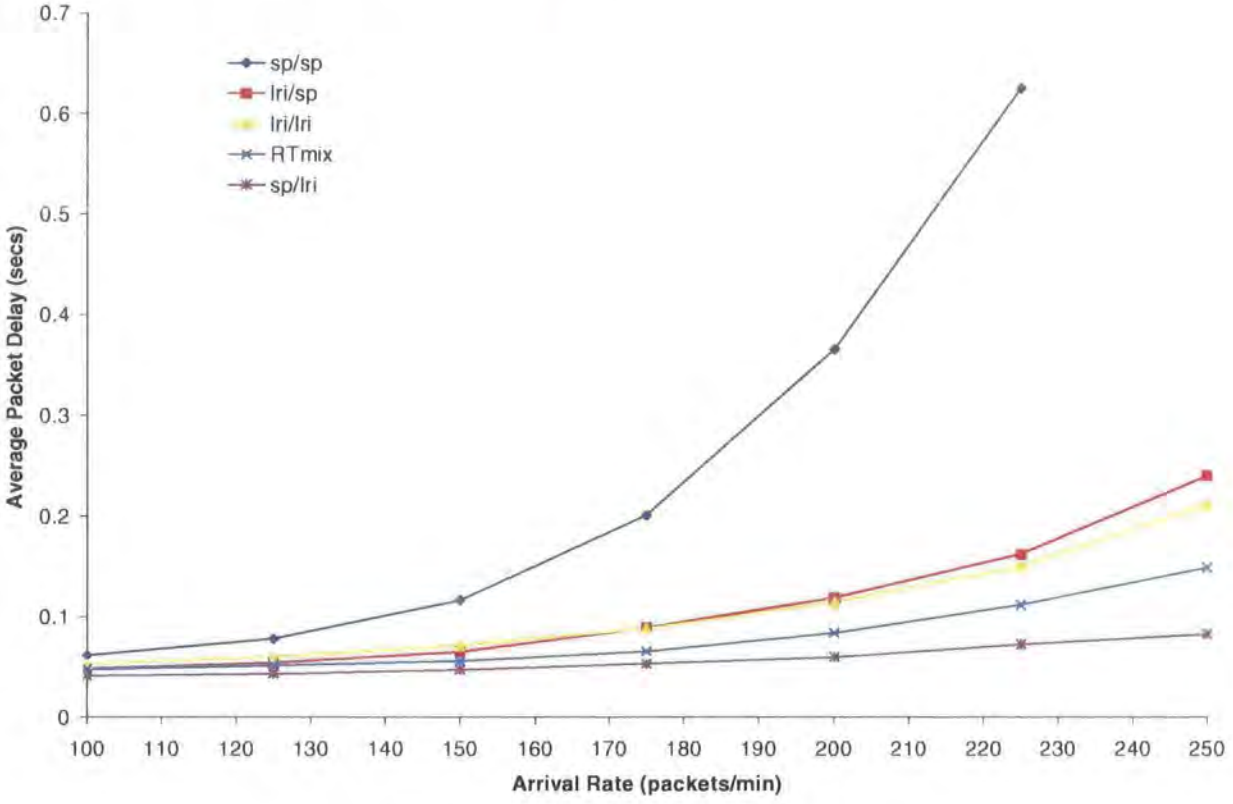


Figure 4.8 - Average NRT traffic delay, mixed traffic.

As observed from the graph, using shortest path routing for both traffic types yields the highest NRT average packet delay. Using learning automata to route the RT traffic improves upon shortest path routing despite the fact that learning automata yield lower blocking probabilities and therefore less spare resource to the NRT. This is because the load balancing behaviour of the automata eases resource consumption along the shortest paths. Thus, adopting automata for the routing of the RT traffic improves the performance seen to the RT and NRT traffics in terms of reduced blocking probability and packet delays respectively. The best delay performance to the NRT traffic is achieved by the sp/lri trace using shortest path routing for RT traffic and automata based routing for NRT traffic. Shortest path routing of the RT traffic leaves maximum spare resources to the NRT, and the NRT automata are able to learn where this capacity is located and make use of it, not necessarily along the shortest paths. The RTmix scheme yields midway delay performance although will deteriorate when the traffic matrices of the two traffic types are significantly different, such that it is impossible to satisfy both traffic demands with one probability vector. However, adopting a common algorithm for both traffic classes means that there will be less storage and processing overhead than using a discrete algorithm for each class.

We repeated the above simulations for a traffic mix with more bandwidth exclusively available to the best-effort traffic. To achieve this, we ran the same simulations as before, but the NRT traffic had access

to an additional (exclusive) 25kbit/s at each node. Thus, NRT traffic had access to a capacity of at least 25kbit/s at each node. Once resource reservation traffic is introduced to the Internet, it is likely that only a small fraction of resources will be reserved explicitly for real-time applications such as video and voice, so that traditional NRT services such as TELNET, FTP etc.. remain largely unaffected. In Figure 4.9, we show the average delay of all the above routing algorithms plotted against nodal arrival rate for the above situation.

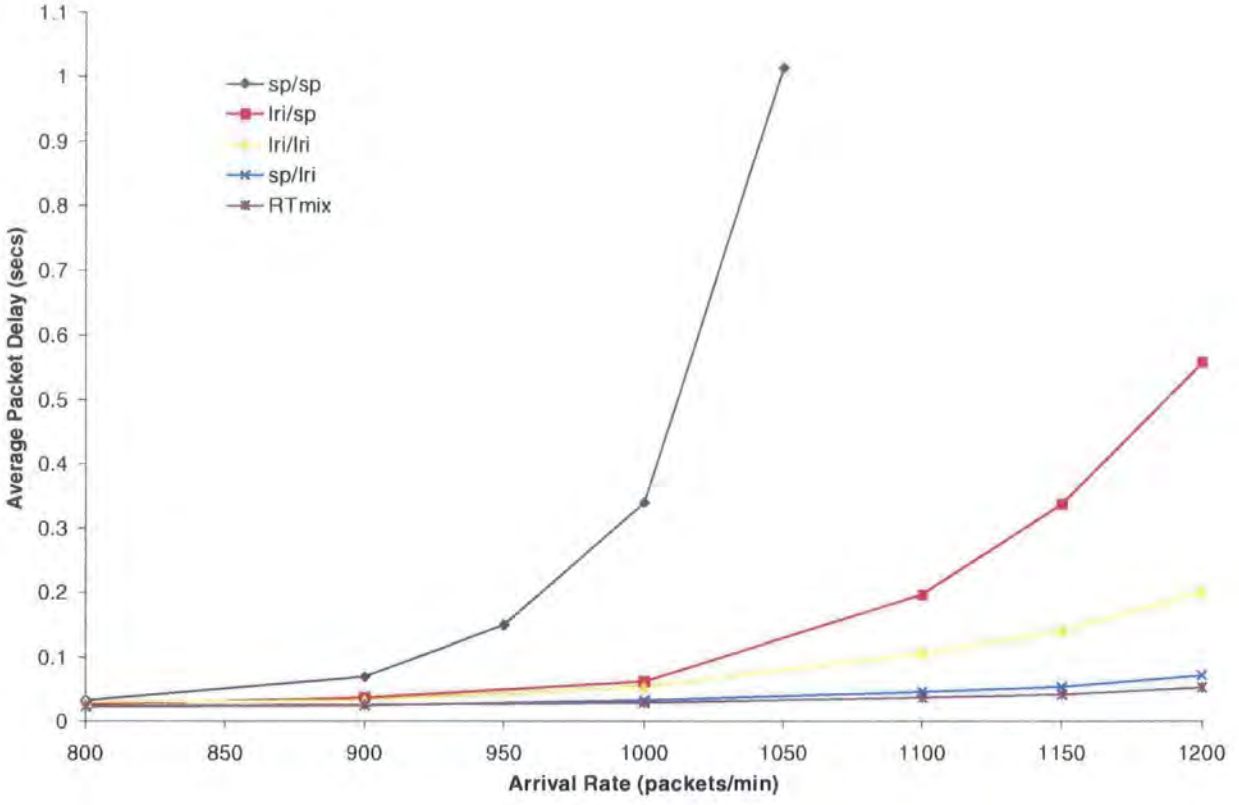


Figure 4.9 – Average NRT delay, mixed traffic, 25kbit/s NRT only.

Due to the increased bandwidth solely for the use of best-effort traffic, the performance of the best-effort traffic in terms of delay is less susceptible to the specific real-time routing algorithm. The main difference here is that the RTmix scheme produces lower delays than the sp/lri scheme, since the RTmix algorithm can produce good routes without the overhead of the control packets of the lri schemes which add to the delay of the data packets. Thus, when the traffic distributions are similar for the NRT and RT traffics, the RTmix routing scheme which uses the RT automata probabilities to also route the NRT can produce low delays to the NRT traffic across a range of traffic mixes. For the final mixed traffic experiment, we have maintained the previous set-up while altering the traffic distribution for the NRT traffic. Specifically, we have used the uneven traffic distribution from Chapter 3 for the NRT traffic generation (see Appendix C). In addition, we have reduced the bandwidth explicitly reserved for the NRT from 25kbit/s to 5kbit/s for node E only in the 10 node network. (Figure 3.2, Chapter 3). We plot the average packet delay against nodal arrival rate for the five routing schemes in Figure 4.10.

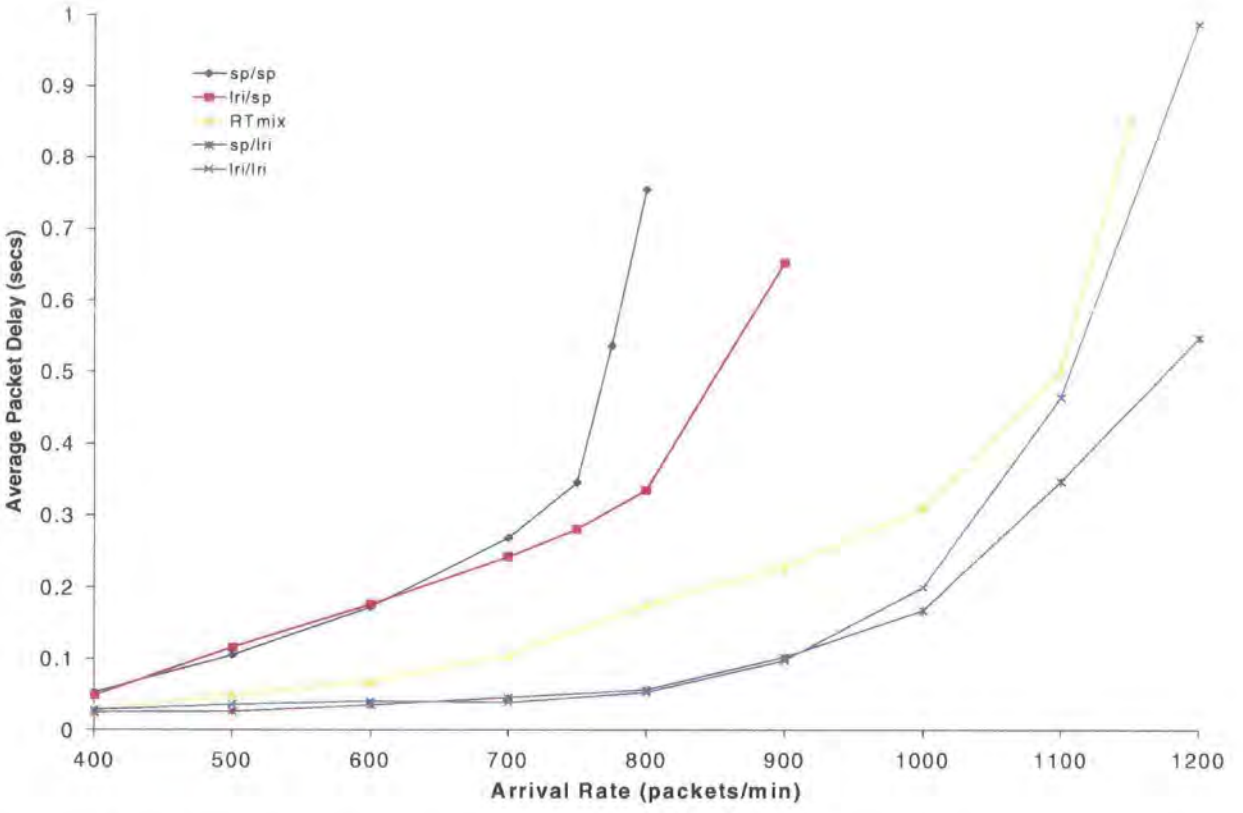


Figure 4.10 - Average NRT delay, mixed traffic, 25kbit/s NRT only, different RT and NRT traffic distributions.

From Figure 4.10, we see that the relative performance of the RTmix scheme has decreased since routes which are preferable for the RT traffic are not necessarily advantageous for the NRT traffic due to their different traffic distributions and their perceived resource availability in the network. Those routing schemes that use shortest path routing for the NRT traffic are particularly bad since a large proportion of the NRT traffic flow passes through node E, which is capacity limited from the NRT perspective.

For all of the previous experiments, the RT nodal arrival rate has been maintained at 25 calls/min. The quality of routes produced by the RTmix scheme will depend on the values of the probabilities for the RT automata, these depending on the relative congestion of the RT traffic. We have used the RT automata probabilities to route the NRT traffic in an environment containing only NRT traffic to gauge the quality of the routes suggested by the RT automata under various RT traffic arrival rates. The simulation set-up is the same as for even traffic demands on the 10-node network as in section 4.4. In Figure 4.11, we plot average packet delay against nodal arrival rate for the case when the RT automata probabilities route the NRT traffic, for RT nodal arrival rates of 5, 25 and 80 calls/min. We compare these traces with the original LRI automata and shortest path traces obtained at the start of section 4.4. In Figure 4.12, we compare the average length of the paths travelled by the NRT packets, which gives insight into the length of the paths that the RT automata are converging to. 90% confidence intervals are shown.

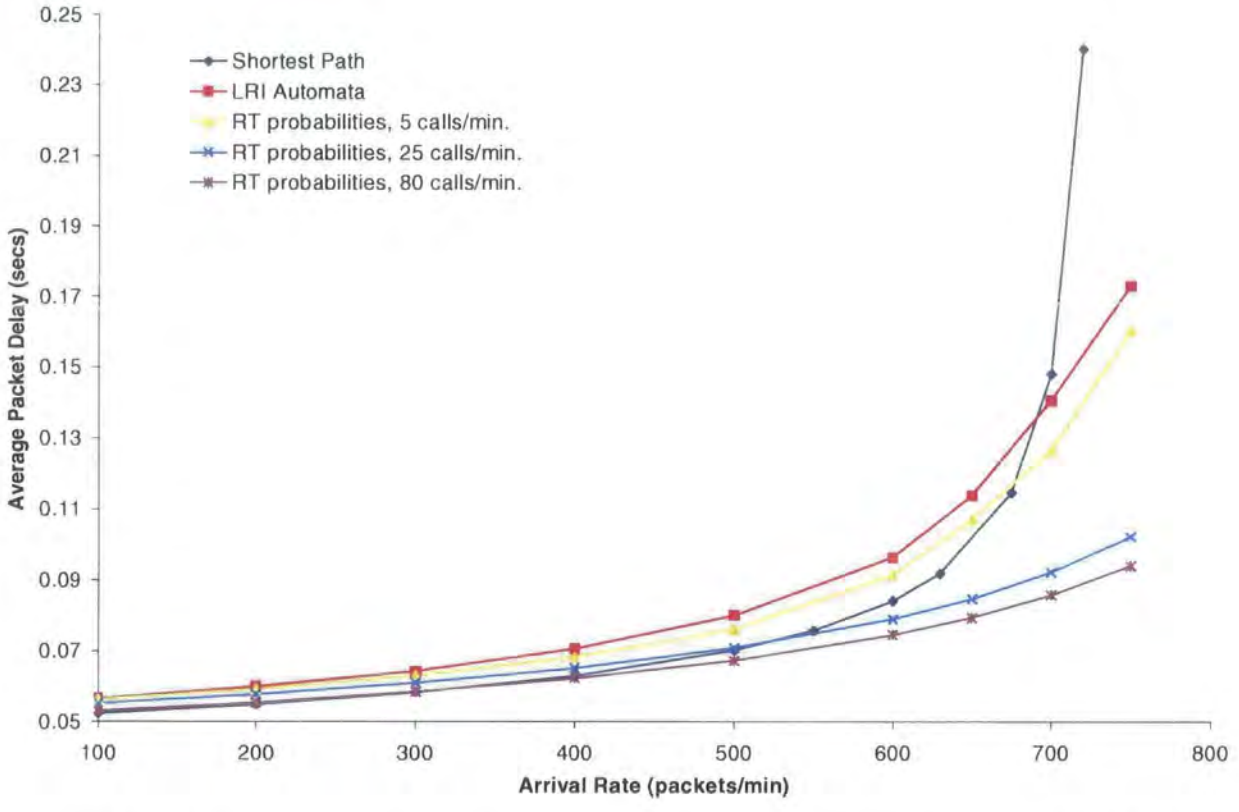


Figure 4.11 – RT probabilities route NRT traffic, various RT arrival rates, average NRT delay.

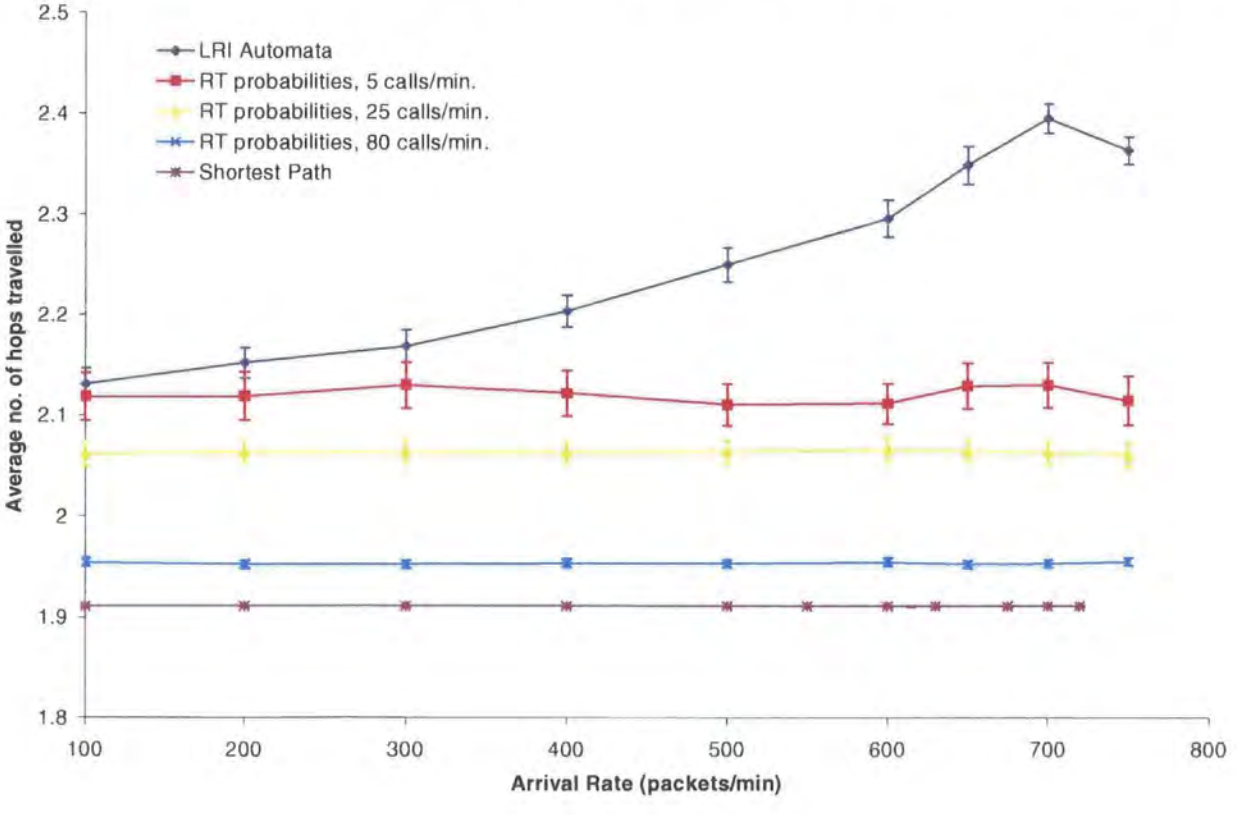


Figure 4.12 – RT probs. route NRT traffic, various RT arrival rates, average NRT path length.

When the arrival rate of the RT traffic is 5 calls/min, there is little contention in the network and the RT automata will converge to arbitrary routes. This can be observed in Figure 4.12, where the NRT packets travel along longer paths, the confidence intervals being high since different simulation runs can result in very different paths being selected by the RT automata. Thus, at low RT arrival rates, the probabilities of the RT automata may not provide a very good load splitting strategy for the NRT traffic. At the higher arrival rate of 25 calls/min, the RT automata start to use a load splitting strategy and the RT probabilities are now a much better indicator of how the load should be split for the NRT traffic. The average path length travelled by the NRT traffic as shown in Figure 4.12, has been reduced indicating that the RT automata are splitting the load over shorter paths. As the RT arrival rate is increased to 80 calls/min, the level of contention is extremely high, so that the RT automata will tend to split the load over the shortest paths only. This results in a further reduction in the NRT average packet delay and path lengths. Using the RT automata probabilities to route the NRT traffic results in a reduction in average delays over the NRT (LRI) automata routing scheme for all three values of RT arrival rates. This is the case since the NRT (LRI) automata communicate finite size control packets to reach estimates of the end-to-end delay for each source-destination pair, and the control packets in the queues add to the delay of the forward path data packets. In addition, it can be seen that the NRT (LRI) automata are tending to converge to the use of longer paths, and constraining the NRT automata to use shorter paths via a tighter hop-count constraint on the length of paths travelled by data packets should bring a significant performance improvement.

4.6. Summary

In this chapter, learning automata have been considered for the routing of non-real-time (NRT) or best-effort traffic. In order to enable automata based datagram routing, it is necessary to introduce a small control packet sent between nodes, such that nodes can form estimates of the end-to-end delays. If automata source routing were adopted, these control packets would not be required. In a mixed traffic environment containing resource reservation and best-effort traffics, it was found that the performance to the best-effort traffic in terms of average packet delays, is dependent on the traffic matrix and routing algorithm used by both traffic types. The RT automata probabilities were found to provide effective routing decisions to the NRT traffic when the traffic matrices of both traffic types are similar, and there is some contention for the RT traffic such that they converge to a load splitting strategy.

Until now, we have focused explicitly on unicast routing. Multicast routing is the process of setting up a distribution tree for shared communication by a multicast group (set of receivers and sources), and can result in significant savings in bandwidth and number of communication messages over multiple unicast set-ups. In the next chapter, we introduce the multicast routing problem and explain how automata may be used to construct multicast trees in dynamic environments to minimise some performance index such as packet delay at the receivers or the total number of nodes in the distribution tree(cost).

Chapter 5

Learning Algorithms for Multicast Routing

5.1. Introduction

In previous chapters, we have examined the use of learning algorithms for providing adaptive unicast routing in future integrated services networks. Here, we extend the idea by examining the use of learning algorithms to 'grow' multicast trees in an environment where receivers may join and leave the trees dynamically. Our specific interest is to examine learning automata for growing shared multicast trees in dynamic best-effort environments. The aim is to show how the automata may enable the minimisation of some cost function such as packet delay at the receivers or the total cost of the resulting trees, with or without (propagation) delay constraints. Our particular interest is how automata may grow such trees in a totally distributed implementation whilst requiring very little information regarding the global network state.

5.2. Multicast Routing

There is a pressing need to consider enhanced communication protocols to deal with multipoint (or group) applications. A multipoint application can be defined as one that involves more than two users that wish to exchange information. This set of users is usually referred to as a group and all members of the group will typically share a common identifying multicast address [135]. Multipoint or multicast communication modes can actually be thought of as a generalisation of both unicast and broadcast communications. Typical multicast applications include updates to replicated databases, command and control systems, audio/video conferencing, distributed games and distributed interactive simulation. Figure 5.1 shows the basic principle of multicasting. Here, a source S is sending to three receivers, R1 through R3. In the unicast approach, we would send each packet generated by the source 3 times, once for each receiver. Alternatively, with a broadcast approach, every node in the network is forced to receive a copy of the packet even if a node doesn't necessarily want to receive packets. For multicasting, one packet is sent by the source to the intermediate node A, where the packet is replicated and sent to each of the three receivers. In this way, only one copy of the packet traverses each link in the network. Efficient multicasting is a fundamental issue for the success of group applications.

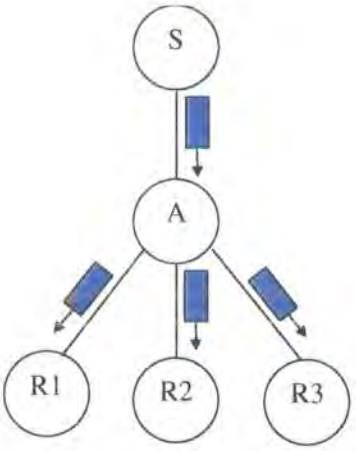


Figure 5.1 – Simple Multicast Tree

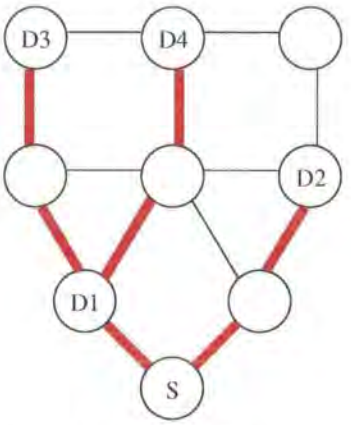
The use of multicast as shown in Figure 5.1 can present significant savings in bandwidth usage over unicast and broadcast transmissions, particularly for large groups. In addition, multicast routing can also take advantage of LAN (Local Area Network) multicast capabilities to reduce the overhead of group communication on a shared medium. Multicasting is considered as a critical issue within the Internet Engineering Task Force (IETF) and future routing protocols are likely to be designed with multicast in mind.

In this chapter, novel ways of setting up multicast trees are considered. These techniques utilise distributed learning algorithms to grow and adapt the multicast tree over time in response to changing group membership and traffic patterns. The chapter is organised as follows. In section 5.3, an overview of traditional multicast forwarding algorithms is given. In section 5.4, we describe our application of learning algorithms to the forwarding problem and in section 5.5, we describe the simulation model used for evaluation. In section 5.6, we provide results of the technique and compare it to more traditional mechanisms and in section 5.7, we give our conclusions and a summary.

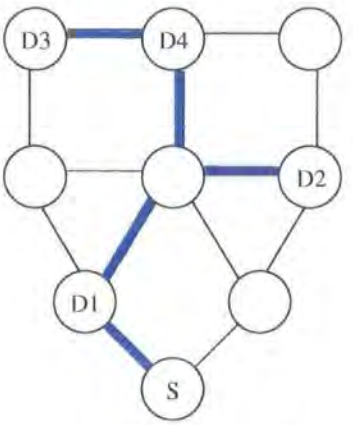
5.3. Traditional Multicast Forwarding Algorithms

In the general case, a multicast session may involve multiple sources sending to multiple destinations and is known as the 'multipoint-to-multipoint' or 'many-to-many' multicasting problem. A special case is the instance of just one source sending to a batch of receivers and this is known as the 'point-to-multipoint' or 'one-to-many' multicasting problem. For the one-to-many problem, a routing algorithm constructs a source routed tree spanning all the receivers. For the many-to-many problem, algorithms may construct source specific trees where a one-to-many multicast tree is created for each source, or it is possible to construct a 'shared' tree where a single tree is set up which is shared by all sources. Multicast routing algorithms generally belong to two categories. The first category are shortest path trees (SPT). Shortest path trees use traditional unicast routing algorithms like Dijkstra and Bellman-Ford algorithms to construct a source routed shortest path tree to the group members. Examples of protocols that create shortest path trees are DVMRP [136], MOSPF [137], and PIM-DM [138]. The second routing category is

known as 'Minimum Steiner Trees' (MST). Here, given some subset of nodes in a graph that must be joined together, the minimum Steiner problem is that of constructing the minimum cost tree which joins all these points, where cost is usually defined as the total number of links or nodes in the tree. This problem is known to be NP-complete [139], although various heuristics exist with worst case proven bounds [140]. In Figure 5.2, for a source S sending to 4 receivers D1 through D4, we show the shortest path tree and the minimum Steiner tree respectively. The example is taken from [141].



(a) Shortest Path Tree. Total cost = 7, maximum path length = 3, average path length = 2.25.



(b) Minimum Steiner Tree. Total cost = 5, maximum path length = 4, average path length = 2.75.

Figure 5.2 – Examples of a Shortest Path Tree and a Minimum Steiner Tree.

Shortest path trees minimise (propagation)delay/path length (from source to receivers) at the expense of tree cost whilst minimum Steiner trees minimise cost at the expense of delay. Thus, the Steiner tree has some routes taking slightly longer paths in order to maximise sharing potential thereby minimising overall tree cost. Typical Steiner tree heuristic algorithms are computationally expensive (see [140] for overview) and usually require a centralised computation utilising knowledge of the link costs/delays for the entire network. A popular Steiner tree heuristic is the Kou-Markowsky-Berman (KMB) [142] algorithm. Here, a shortest path set between the group members is formed and a minimum spanning tree is taken of this set. The approximate Steiner tree is then obtained by achieving the shortest paths represented by edges in the minimum spanning tree. The KMB algorithm has a worst case computational complexity of $O(G N^2)$ where N is the number of nodes in the network and G is the size of the multicast group [143], and assumes that each node has access to the link cost/delays for the entire network. Additionally, for dynamic environments where receivers are joining and leaving the group, the Steiner approximation must be recomputed each time the tree changes which can incur considerable computational overhead in addition to disrupting flows to current members of the multicast session. Steiner tree algorithms are therefore best suited to slowly changing environments where we do not have to compute the optimal tree very frequently (e.g. layout of circuit boards etc). To avoid the complexity of a Steiner tree approach when constructing shared trees, a 'centre-based tree' may be used. Here, the idea

is to construct a single tree which is shared by all members of the group, where the tree is routed at a topological centre. The 'core-based tree' (CBT) algorithm, and the 'Protocol Independent Multicast – Sparse Mode' (PIM-SM) introduced by Ballardie [144] and Estrin *et al.* [138] respectively, are examples of a centre-based tree technique and represent totally receiver oriented approaches meaning that receivers (rather than the sources) are entirely responsible for joining the group. The basic idea is for receivers to send a join message towards the centre(s) until the message reaches a member of the tree. Once a join message reaches a member, a join acknowledgement is sent to the requesting receiver which will then receive any data packets sent to the specific multicast group address. Also, a node can send data to the group without being a member simply by forwarding packets to the nearest core. In general, a centre-based tree will not be optimal for any one sender but may well be an adequate approximation for all of them. In Figure 5.3, we present a summary of multicast routing algorithms for source specific and shared trees.

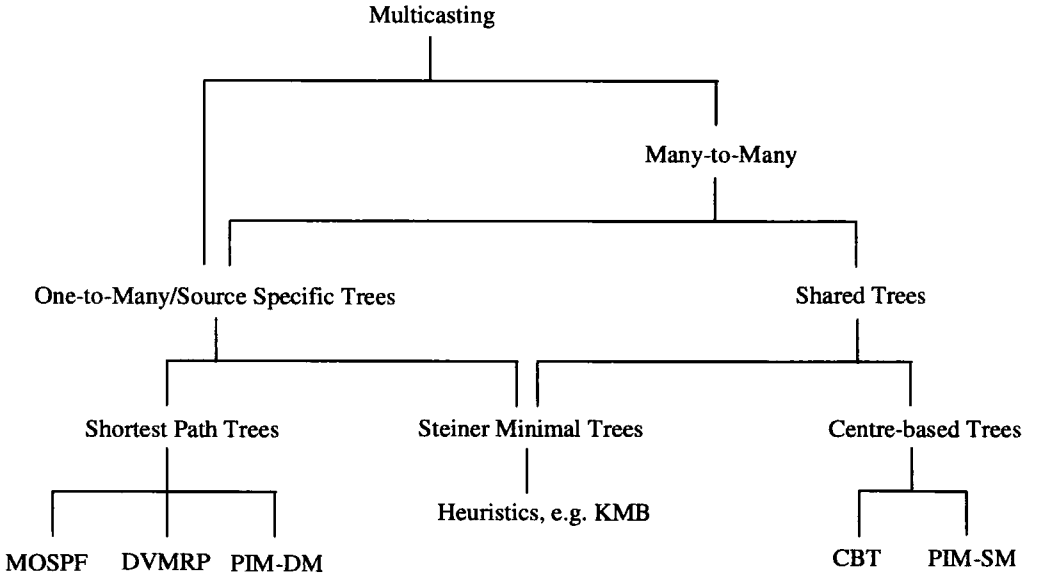
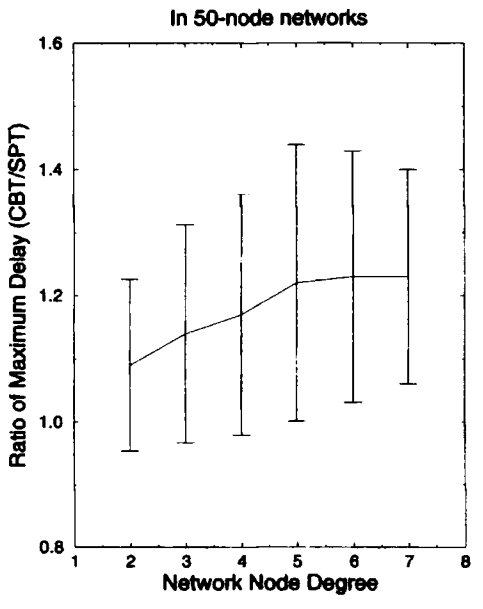


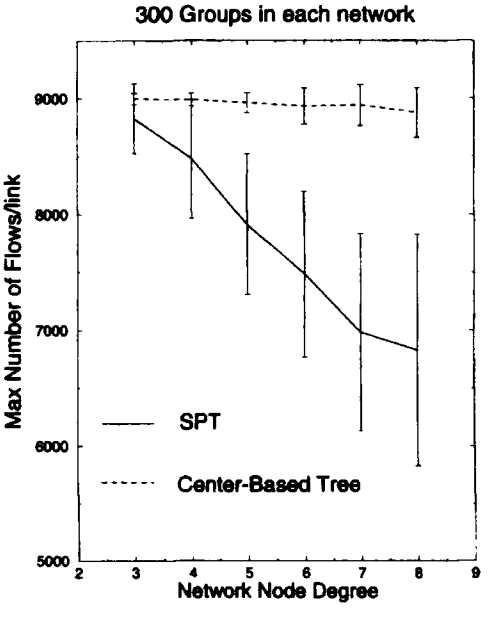
Figure 5.3 – Summary of Multicast Routing

The three approaches (shortest, Steiner and centre-based) described for multicast tree construction involve relative trade-offs between the delay, cost and traffic concentration characteristics of the resulting tree. Source routed shortest path trees achieve minimal delay performance since paths, by definition, are shortest path ones. Steiner minimal trees (SMT) minimise cost at the expense of delay. Between these two extremes, there are a spectrum of different types of trees offering different trade-offs. In addition, different routing algorithms will use different strategies to place the routes and could result in differing degrees of traffic concentration. Centre-based trees also fall between these two extremes. An extensive simulation study of centre-based trees carried out in [145] has shown that the shared tree cost is slightly lower than that of shortest path trees at the expense of delay performance. Specifically, if the centre is located at a group member and as many trees as group members are considered to locate the centre, then it is shown in [145] that delays are close to 20% larger than for shortest path trees, and tree cost is about

10% lower than that of shortest path trees. Additionally, the same study showed that traffic concentrations could be up to 30% greater than shortest path trees using the same core placement strategy as above. Traffic concentration occurs for the centre-based trees since sources share links in the distribution tree, particularly as we get closer to the core. In Figure 5.4 (a) and (b), we show graphs of SPT and CBT delays and traffic concentration respectively. The graphs are taken from [138], where multiple simulations were carried out on randomly generated graphs of various node degrees. The first graph shows the ratio of CBT maximum delay to SPT maximum delay measured across 500 different 50-node graphs for increasing node degree. The second graph shows the maximum number of flows per link for 500 different 50-node graphs with 300 active multicast groups, each with 40 members, 32 of which were sources.



(a) SPT and CBT delays.



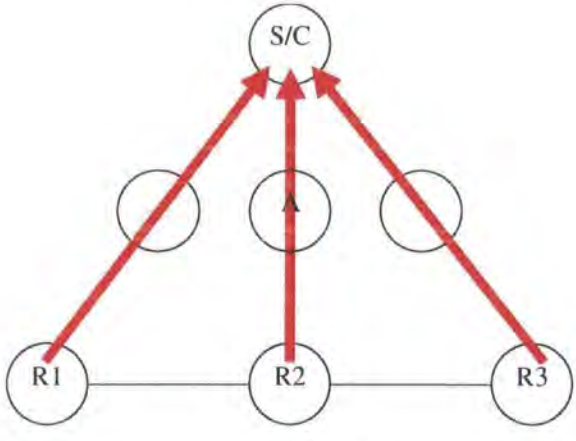
(b) SPT and CBT traffic concentration.

Figure 5.4 – Comparison of SPT and CBT.

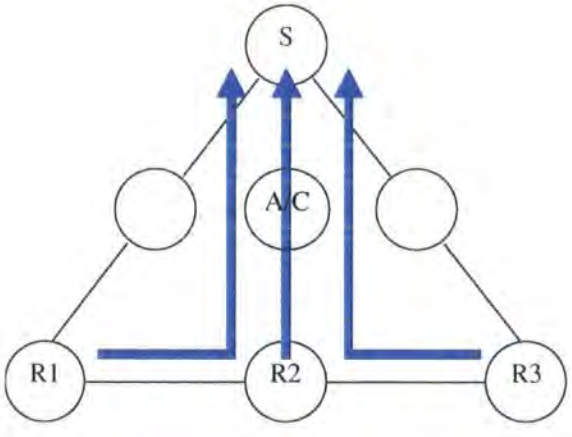
The main advantage of the centre based tree approach is from a scalability (and maintenance) perspective since a router maintains state information for each group, not for each (source, group) pair as for shortest path trees. Specifically, if there are G multicast members and S sources, centre-based trees scale as O(G) whereas source-specific trees scale as O(S.G) since nodes must store state for each source sending to each group member. Thus, for multicast groups with many members and sources, centre-based shared trees cut down on the amount of state to be stored. Although a shared tree does not strictly have to have a centre, doing so facilitates several tree management functions. The centre acts as a reference point so that a receiver wishing to join the group merely has to send a join request towards this centre. Similarly, if a source starts transmitting to a session, it simply has to transmit towards the core to guarantee that receivers will receive the transmission. The drawback of a centre based tree is that the centre must be

contained within the distribution tree which may limit how close we may get to the optimal shared tree in addition to the traffic concentration effect described above. The centre-based tree problem may be split into two parts: the centre selection problem and the route selection problem. The centre is typically selected based upon some heuristic depending on member locations in the network. Choosing a centre to optimise some cost function is well known to be an NP-complete problem [146], and has also been examined in [147]. Wall [148] showed that a topologically centred tree gives a worst case delay bound of twice that of a shortest path tree and proposed some heuristics for core selection based on some measure of the distance (average, peak etc..) from the core to the group members. The route selection process involves receivers selecting the appropriate paths to the centre(s) of the group. For example, for CBT, receivers route on the shortest paths to the core.

One of the outstanding problems with multicast tree formation is how to cope with the dynamics of the group membership as receivers join and leave the multicast group(s). The migration of cores has been proposed [149, 150] to attempt to maintain minimal delay in the face of changing group membership. Consider the multicast trees shown in Figure 5.5 for a single source S sending to three receivers, R1 through R3. Assume now that we operate a centre-based tree technique where the receivers route join requests on the shortest paths to the core. If we locate the core at the source, then we will achieve the minimum delay (shortest path) tree as shown in Figure 5.5 (a). If we now locate the core at node A and assume that the source sends data on the shortest path to the core, we can see that we will achieve the minimum cost tree as shown in Figure 5.5 (b).



(a) Shortest Path Tree. Total cost = 6, maximum path length = 2, average path length = 2.



(b) Minimum Steiner Tree. Total cost = 4, maximum path length = 3, average path length = 2.67.

Figure 5.5 – Minimum delay and cost trees.

Thus, we can see the benefit of moving a core to achieve minimum delay or cost performance respectively. There may be considerable overhead associated with moving a core however since nodes will need to exchange information to select a new core position, and a trade-off must be reached between acceptable delay/cost performance and the frequency of core migrations. Since a centre based tree is

routed at the centre, moving the centre will also result in disruption to the members taking part in the multicast session. An alternative method would be to have one or more cores in the network but use a distributed algorithm at the receivers so that they can choose to route to a different core (not necessarily the closest) or choose alternate paths to a particular core. Thus, we are interested in the route selection part of shared trees, and assume that the core(s) has been administratively located somehow. For example, suppose that a single core has been placed at node A to achieve minimum cost, but we wish to minimise the delay to the receivers instead. Then all we require is for receivers R1 and R3 to route on the longer paths to the core which pass through the source rather than on the shortest path to the core. Receiver R2 continues to route on the shortest path to the core since this is also the minimum delay path for R2. This is shown in Figure 5.6.

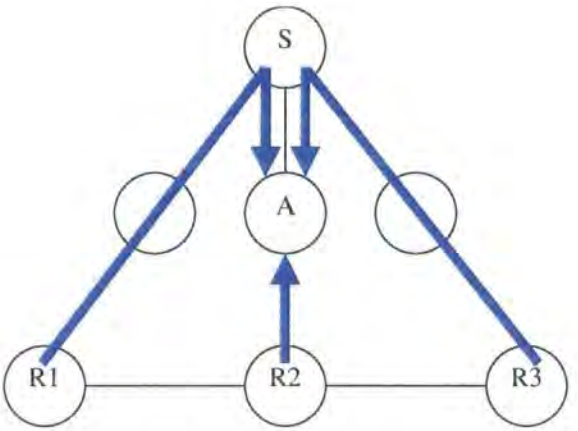
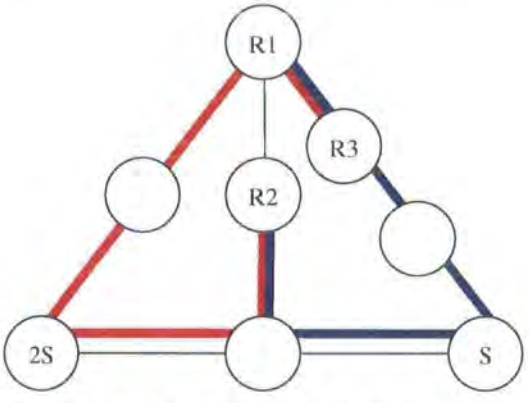


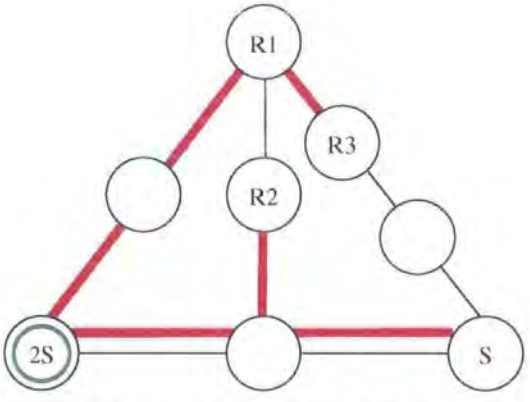
Figure 5.6 – Alternate Path Based Tree.

Thus, the application of a distributed learning process can potentially improve delay/cost performance by letting receivers learn of appropriate alternate paths to the core(s). For the minimisation of delay, receivers will learn to send to the cores closest to the most dominant sources sending to the multicast group. In a similar way, assuming a single core had been placed at the source to minimise delay, the use of alternate paths by the three receivers could lead to the minimum cost tree as shown in Figure 5.5 (b). Thus, we retain the scalability advantages of a shared tree approach whilst utilising distributed learning algorithms to minimise either the average received packet delay or the total tree cost. The above examples only show one source in the network. When there are multiple sources sending to a multicast group, a shared tree cannot be expected to replicate the low delay of a shortest path tree since joining the tree close to one source may result in being further from other sources. Consequently, we look to automata to route towards the dominant sources in the network to reduce the received packet delay. For example, consider the network shown in Figure 5.7. Here, we depict two sources sending to a group, S and 2S, whereby 2S is assumed to send to the group at twice the rate of S. Consequently, the delay from the receivers R1 through R3 to 2S has twice the weighting of that to S. Link delays are 1 and 2 units for S and 2S respectively. In Figure 5.7, we show the shortest path tree (one for each source), one instance of

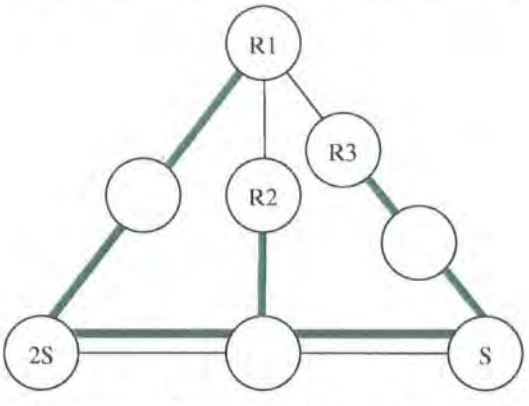
the optimal delay centre-based tree and the optimal delay shared tree respectively. For the centre-based tree, the node with the core is marked by a concentric circle. We document the total receiver delay and the total link cost of each tree. We see that we could arrive at the optimal shared tree solution of (c) if the core were at the same location as for the centre based tree, but receiver R3 sends a join request along the longer alternate path through S to the core.



(a) Shortest Path Trees, delay = 21, cost = 8.



(b) Centre based tree, delay = 25, cost = 6.



Optimal delay shared tree, delay = 24, cost = 7.

Figure 5.7 – Example trees, multiple source case.

Although the above examples show only one core or centre in the network, the use of multiple cores can have several advantages over the single core case including increased resilience and reduced traffic concentration since the traffic can be spread amongst the cores. Even if a network only supports shortest path based unicast routing, the use of multiple cores allows us to minimise delays by having receivers send join requests to the core nearest to the dominant sources in the network. Learning is applicable here so that the set of distributed receivers may learn to send join requests on the shortest path to the relevant core. In [141], it is thought that multiple cores may also help to achieve different Quality-of-Service (QoS) levels since each core may be associated with a different shared tree providing a different QoS level, and receivers would select the appropriate core to achieve their required QoS. Also, a single centre based shared tree may not be able to meet an upper bound on the delay between any source and destination, so that multiple shared trees with corresponding centres could be designed to meet any

required delay constraint. The use of learning algorithms for creating source-specific and shared QoS-bounded trees is examined in Chapter 6.

5.4. Learning Algorithms for Multicast Routing

In the previous section, we saw how the application of learning to form alternate paths may be beneficial for growing multicast trees to minimise some performance index such as received packet delay or total tree cost. Here, we describe in detail how learning automata have been used to achieve this. We assume a centre-based shared tree environment where receivers are responsible for joining a group. In our description below, we assume that a backbone of cores has already been constructed, and that all nodes have knowledge of which nodes in the network are cores, this knowledge likely having been obtained using a 'bootstrap' process in a real network. Note that with the receiver oriented join process which we adopt, receivers do not require knowledge of the location of other group members, since this would require a communication process between the nodes, perhaps via flooding, to enable group member location. We believe that such a flooding process will not scale to large and/or dynamic multicast environments.

5.4.1. Source Routing Automata

This algorithm works as follows. Each potential receiver node contains a learning automaton for each multicast group in the network. The actions of each automaton represent the probability of sending a join request on the shortest path to each of the cores in the network. When a receiver/node decides that it wants to join a group, the learning automaton stored at that node chooses a core to send the join request to according to the probability distribution of the automaton. The address of this core is then placed in a field in the join packet and the packet is forwarded to the next node on the shortest path to the chosen core utilising underlying unicast routing algorithms to do so. The join request is source routed in that, intermediate nodes simply route the join request to the core that was initially chosen using the field in the join request packet. That is, intermediate nodes do not make their own routing decisions (i.e. select an alternative core to the original choice). In this way, we avoid possible routing loops that could be formed when independent routing decisions are made as in 'hop-by-hop' routing. Once the join request reaches a member of the distribution tree, the requesting receiver should receive an acknowledgement informing it of a successful join operation. The receiver will then receive any data packets sent to the multicast group address. Figure 5.8 shows the principle. Node 6 wishes to join the multicast group. Since there are two cores, the automaton located at node 6 will have two actions, which represent the probability of sending the request on the shortest path to C1 and C2 respectively. For example, if node 6 chooses to send the request to C1, the request will be forwarded on the shortest path to node 1 which will accept node 6 as a new child and send back a join acknowledgement to node 6.

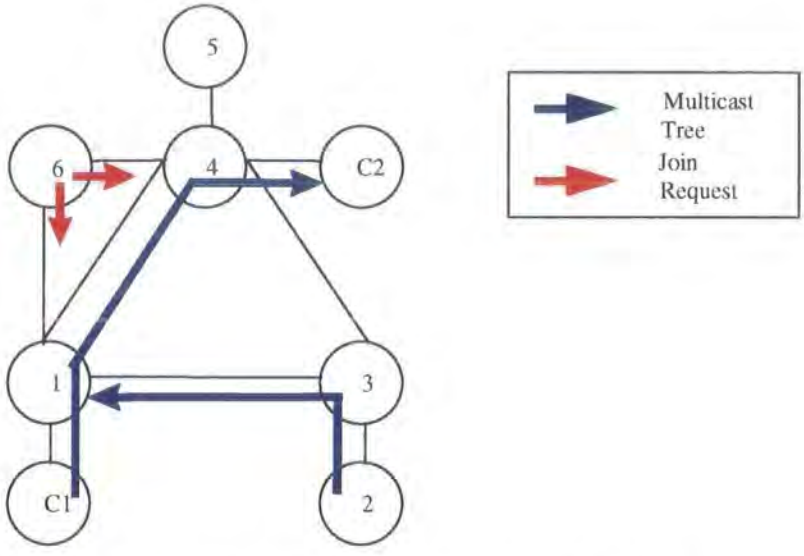


Figure 5.8 - Multicast tree and join behaviour

Once a (leaf) receiver leaves the multicast group, the average packet delay is calculated over all packets received by the receiver during that session. It is assumed that data packets have a time-stamp field to make this possible. This delay is then transformed into feedback to the learning automata so that the probabilities of sending to each core can be updated. The transformation used is identical to that used for the packet switched automata experiments in Chapter 4, where the measured session delay is transformed into the region (0, 1) using the minimum recorded session delay so far. (equation 4.5). As for the delay experiments in Chapter 4, S-type automata are used since delay is a continuous variable. In this way, as receivers continually join and leave the multicast group, the distributed set of learning automata will learn to send join requests to the appropriate cores in order to minimise the average packet delay.

In terms of storage requirements, the proposed algorithm scales as O(kG) where k is the number of cores (per group) and G is the number of groups whereas standard CBT scales as O(G). This is due to the fact that routers/nodes must now store the probability of sending a join request to each of k cores in addition to the address of the cores themselves, whereas for standard CBT, the router simply stores the address of the nearest core. We know that a shortest path approach scales as O(S.G) where S is the number of sources. Thus, the source routing automata scheme will still scale well, particularly for cases where there are a large number of groups and sources.

5.4.1.1. Avoiding Routing Loops

Join request set-up packets record the route taken so that a unicast loop resulting in a join request visiting some node twice can be easily spotted and prevented. Since routers can choose to route a join request to different cores however, a number of constraints have to be enforced in order to prevent the formation of multicast routing loops, which can be disastrous for multicast routing protocols. As an example of a routing loop, consider the network in Figure 5.9 below. Here, the nodes 1, 4 and 3 send join requests to cores C3, C2 and C1 respectively. If the three nodes send out join requests simultaneously, the routing

loop 1-4-3-1 is formed and all three nodes will be permanently waiting for a join acknowledgement.

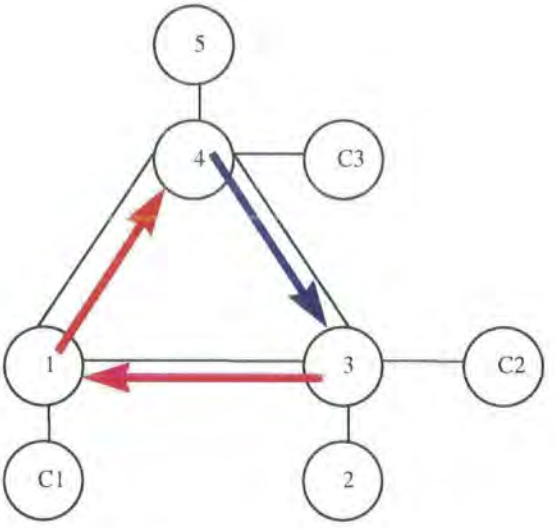


Figure 5.9 - Routing loop formation

To overcome this problem, if a join request reaches a node that has already just forwarded a join request of its own, the intermediate node will send back a 'connect fail' if the destination core of the new request is different to its own. Upon receipt of a 'connect fail' packet, a node will forward the connect fail to any children that it may have and remove them from its database. This simple constraint ensures that routing loops should never occur by requiring that each set-up matches the upstream route already in place on the tree. This mechanism has also been utilised in [94] as a means of avoiding routing loops when utilising alternate path based multicast routing and is also adopted in the next chapter.

5.4.2. Hop-by-hop Automata

For this algorithm, each potential receiver node again stores a learning automaton for each destination group. Now, the actions of each automaton represent an outgoing link on which to send a join request to the group rather than a core to send to. This algorithm scales as O(cG) where G is the number of multicast groups in the network and c is the average nodal connectivity. This is the case since each node stores the probability of each outgoing link rather than a single address of the nearest core as for CBT. With this routing algorithm, automata may choose from all potential routes in the network rather than being constrained to the shortest paths to each of the cores as for the 'source routing' automata described previously. The mechanism for updating the automata probabilities is the same as described for the source routing automata so that the average session delay is transformed into the appropriate feedback using the minimum recorded session delay so far.

5.4.2.1. Avoiding Routing Loops

Like the alternate source routing technique described previously, we require that the alternate route chosen by a node match the route already in place on the tree upstream. However, unlike the source routing technique, there is no way of knowing *a priori* what route will be chosen upstream by the independent learning automata located at each node. Thus, the only way to prevent the formation of routing loops is to only allow one outstanding join request at any one time, thus preventing the merging of join requests. This will only prove a problem when there is a significant probability of more than one join request arriving at a node within close proximity. This should only be the case for extremely dynamic, large multicast environments where receivers are joining and leaving the group very frequently.

5.4.2.2. Minimising Cost

The above description of hop-by-hop automata describes how the distributed set of automata use the average session delay as feedback to the automata so as to minimise the overall packet delay at the receivers. We have also investigated how automata may be used to minimise the overall cost of the tree(s) using the appropriate feedback from the environment. In [151], Waxman used a heuristic which he called a 'greedy algorithm' where receivers route join requests toward the nearest member of the tree such that the join request will reserve the minimum resources in joining the tree. He found that the average cost performance of this heuristic algorithm was within a few percent of the trees created by the KMB algorithm which is known to give overall tree costs close to the minimal Steiner trees. The problem with Waxman's algorithm is that receivers need to have knowledge of the group membership and topology. This could be achieved using member location broadcasts in a similar manner to the mechanism adopted in MOSPF [137], although this would lead to significant receiver discovery overhead. For the feedback to the automata to minimise cost, we use the number of hops a join request has travelled to join the tree. Since join requests record the route travelled to prevent formation of unicast routing loops as described above, we know the sequence of nodes taken in joining the tree and consequently the number of hops travelled to join the tree. Thus, upon receipt of a join acknowledgement, a receiver will extract the hops travelled to join the tree and transform this into the appropriate feedback to the automata using the minimum recorded number of hops to join the particular group so far. We utilise the same square root law transformation as for the delay case above and in Chapter 4. In this way, the distributed set of automata will learn to send join requests to the nearest group members without the need for member location broadcasts.

5.5. Simulation Model

The simulation model closely follows the standard CBT specification [152]. As stated previously, the model has been configured so that the backbone of core routers is set up prior to the start of the simulation. In practice, the protocol has functionality in place to initialise the cores and set up the core connectivity. For the purpose of these simulations however, this is not necessary since the initial core

connectivity is configured by hand. Also, because we are routing to multiple cores, there are a number of cases where routing loops can form as described in the previous section. It is therefore necessary to create another message type which is a 'connect fail' to inform a receiver that the connection cannot be set-up to prevent the potential formation of a routing loop. The other main difference between the simulation and the CBT protocol is that CBT uses 'soft state' mechanisms to time out members of the group. This is achieved by having each child send a periodic 'keep alive' message to its parent. Thus, if a time out occurs then the parent considers the child to have left the group. In the simulations presented here, members (leaf routers) leave the group explicitly by sending 'leave' messages to the parent. This slight alteration will have little impact on the results of any simulation from the view of evaluating whether learning algorithms are a viable option for dynamic multicast tree construction. Using hard state rather than soft state obviates the need for complex timers in the simulation model and reduces the amount of communication traffic thus speeding up simulations.

For the set of distributed automata to 'learn' the appropriate tree, receivers must be continually joining and leaving the group, so that the automata can alter the tree and get feedback on its performance. The problem where receivers join and leave the multicast session is known as the 'Dynamic Multicast' problem [151]. Receivers are modelled as having an average 'on' and 'off' time. When a receiver is 'on', if it is not already a member of the tree, it will try to become so by sending a join request to a core as selected by its learning automaton. When a receiver is 'off', it will try to leave the multicast tree if it is a member and it has no children, and will attempt to leave the tree at the first opportunity (i.e. when all children have left the group) Thus, a node cannot leave the multicast tree if it has children so that routes already in place are 'pinned' (i.e. we do not allow re-routing of the tree). If nodes with children were to leave, this would disrupt the flow to its children who must then re-join the tree possibly via another node. It is important that disruptions are not permitted to occur (unless due to link/node failure), particularly for 'real-time' flows who may have contracted a certain quality of service (QoS) with the network provider. The application of learning automata to the construction of QoS multicast trees is examined in Chapter 6. For the simulations reported, receiver on/off times are drawn from an exponential distribution.

The network models constructed consist of a set of N nodes connected to each other as specified in a connectivity matrix. Each node is modelled by a single FIFO queue with a certain processing rate. All queues have infinite capacity. All link propagation delays are 1ms. Packet sizes are 416 bits.

5.6. Results

To investigate the viability of the learning automata approach to CBT, simulations have been carried out on the 30 node network shown in Figure 3.3 in Chapter 3. We firstly examine results for automata applied to minimising receiver packet delay and go on to examine how learning automata minimise tree cost. Nodal processing rates are set at 50 kbit/s unless stated otherwise.

5.6.1. Single source case

For the single source case, 5 cores are arbitrarily selected (nodes 27, 25, 19, 16, 7) to form a backbone and the source is located at one of the cores (node 25) so that the optimal delay tree occurs when receivers send all join requests to this source/core. For the case considered, 7 nodes are selected as receivers (2, 6, 10, 11, 17, 22, 28) and the average on/off times are 1 minute. Thus, we are simulating a reasonably dynamic environment where receivers come and go relatively quickly. In addition to the two learning based tree construction methods, we have simulated three additional algorithms. Firstly, we simulate source routed shortest path trees by routing all join requests on the shortest path to the source, secondly we route join requests to the nearest core as with standard CBT, and finally we simulate a random strategy where the probability of sending a join request to any core is equal (i.e. 1/number of cores). For the learning automata, LRI reinforcement is used with a learning rate of 0.01 for both source routing and hop-by-hop automata schemes. We compare the average received packet delay at the receivers as a function of time, when the source (node/core 25) sends at a rate of 50 packets/minute to the multicast group. Figure 5.10 depicts sample paths of the average received packet delay for the five multicast tree construction methods considered. The plots represent moving average values averaged over a 100,000 second window.

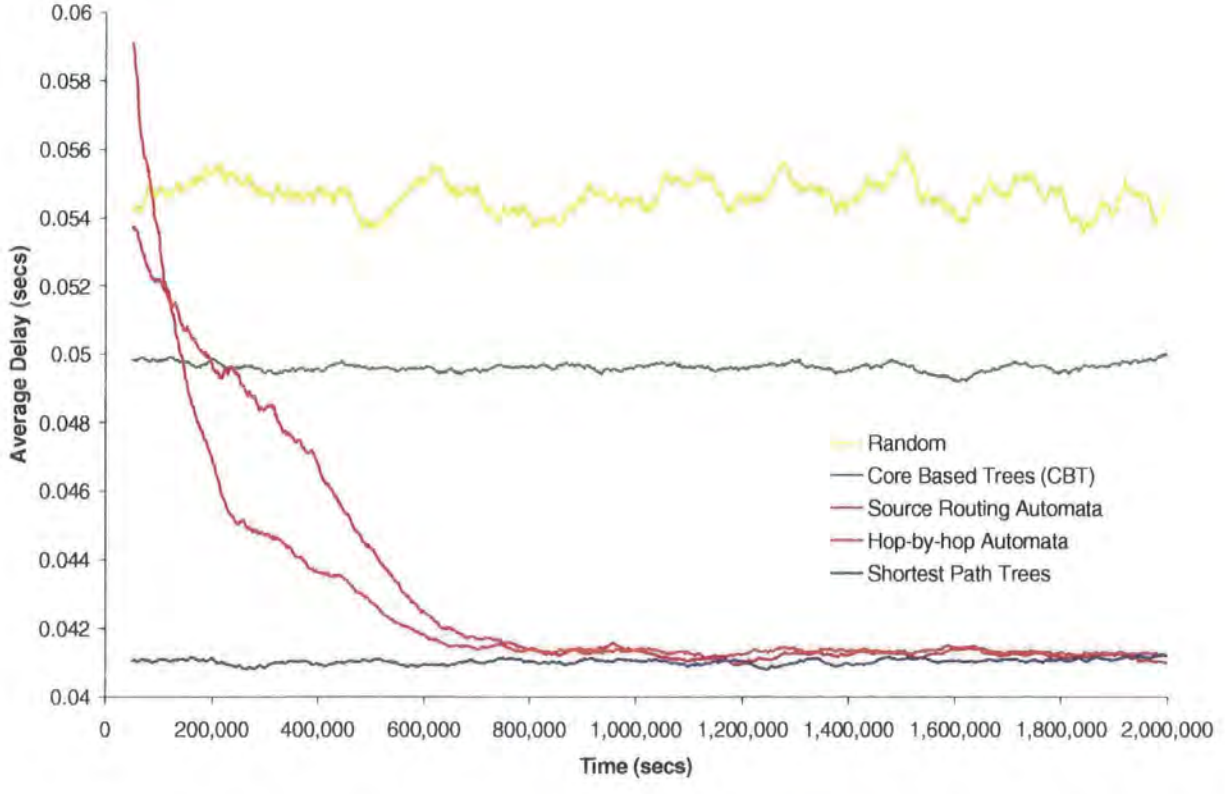


Figure 5.10 - Average packet delay, sparse mode.

The above graph shows how the distributed learning automata, located at the receivers, learn to send join requests to the source (core 25) and thus minimise the average received packet delay. The average packet

delay of both automata algorithms can be seen to gradually reduce to the level of the shortest path tree approach. With the CBT approach, where join requests are sent to the nearest core, the average packet delay can be seen to be around 20% greater than the shortest path trees in this scenario. The performance of CBT relative to shortest path trees will depend on the placement of the cores and the sources and the changing group membership, although [145] has found that CBT produces trees with delays of 20% greater than the shortest path trees on average. The traces also display different variances in delay amongst the algorithms. For the random approach, the variance in delay is high as the tree is continually changing between wide bounds based on the random cores selected. The CBT method produces lower variance in delay as the tree undergoes less radical changes as receivers route to their nearest core. For some members, the core located at the source will be the closest core so they will receive packets with minimal delay. For some other receivers, routing to the closest core will not result in minimal delay, so the average global delay will vary depending on the members of the tree at a particular instant in time. For shortest path trees and the automata in steady state, the dynamic tree undergoes the least transformation as receivers route to a single core (the source), all receivers receiving packets with minimal delay, thus resulting in a dynamic tree with lowest variance in the received packet delay. For reference, the time of 2 million seconds corresponds to 100,000 joins to the multicast tree. Thus, the automata take around 50,000 join requests to converge for this scenario.

5.6.2. Multiple source case.

When there are multiple sources present sending data to the multicast group, there will not be a single optimal action or core to send join requests to as in the previous experiments. Now, sending joins to the tree close to one source could inevitably lead to longer packet delays from another source. Thus, we look to learning automata to find a good compromise by probabilistically splitting the join requests to the relevant cores in order to minimise the average received packet delay. In the following experiment, nodes 18, 17, 20, 19 are cores, nodes 7, 16, 27 are sources and all nodes apart from the cores are receivers. The shortest path tree delay has been approximated by making each source a core in turn and averaging the delay over the three sources and is shown as a constant delay value for reference purposes. For the sources, all packets are sent to the nearest core if they are not currently a member of the distribution tree. In addition to multiple sources, we model the effect of heterogeneous resources in the network by making core 17 have a processing rate of 5kbit/s, 10 times less than all the other nodes. Thus, from a transmission delay perspective, routing a packet through node (or core) 17 is equivalent to routing through 10 other nodes. We expect automata to adapt to this heterogeneity by routing flows such that as many packets as possible do not have to pass through this node. However, a hop count based technique like standard CBT will be insensitive to this node, and we would therefore expect increased average delays. In addition to the source routing and hop-by-hop automata schemes which use average session delay as feedback, we have simulated the hop-by-hop automata algorithm using the average session hop-count of received data packets as feedback to the automata. Thus, the automata in this case will learn to

minimise the number of hops data packets travel to the receivers rather than the actual end-to-end delay. Figure 5.11 shows the resulting average packet delays for this scenario. A moving average window of size 100,000 seconds has been applied to the raw data.

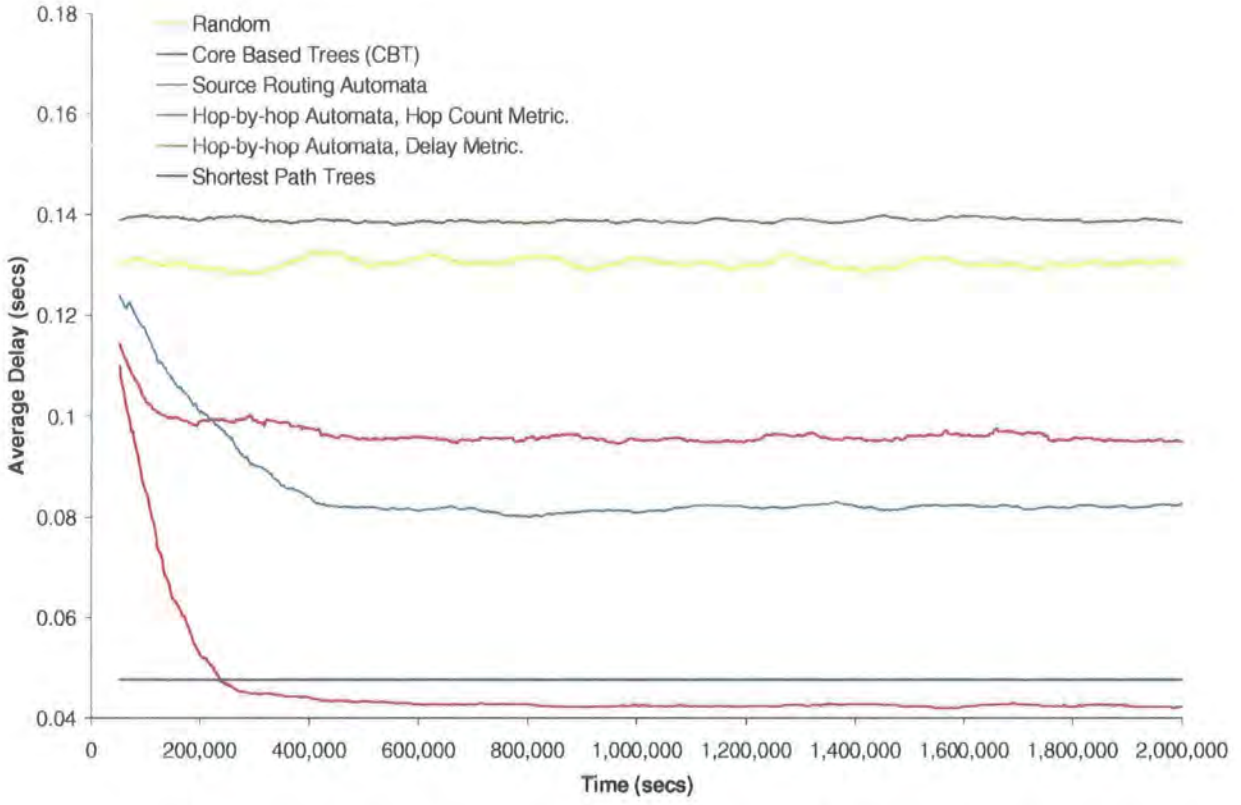


Figure 5.11 - Average packet delay, multiple sources and heterogeneous resource.

The time of 2 million seconds corresponds to just over 200,000 join requests for the hop-by-hop automata. The hop-by-hop automata using a delay metric outperform the shortest path trees in this instance as the automata learn to avoid the capacity limited core. As expected, we see that standard CBT has a considerably larger delay than a learning automata (delay metric) approach in this case, greater than a random strategy in fact as routing to the nearest core here results in a large amount of traffic passing through the capacity limited core. The source routing automata using a delay metric as feedback learn to minimise the delay to some extent but cannot reduce it to the level produced by the hop-by-hop automata since they only store the shortest path to each core, whereas the hop-by-hop automata can effectively choose from all possible paths to minimise the delay. The automata using hop-count as a feedback metric also fail to match the hop-by-hop automata using a delay metric since hop-count gives no indication of the extra delay incurred from passing through the capacity limited core. In Figure 5.12, we show plots of entropy against number of join requests for the delay based hop-by-hop and source routing automata. We see that both routing schemes are converging in around 100,000 join requests.

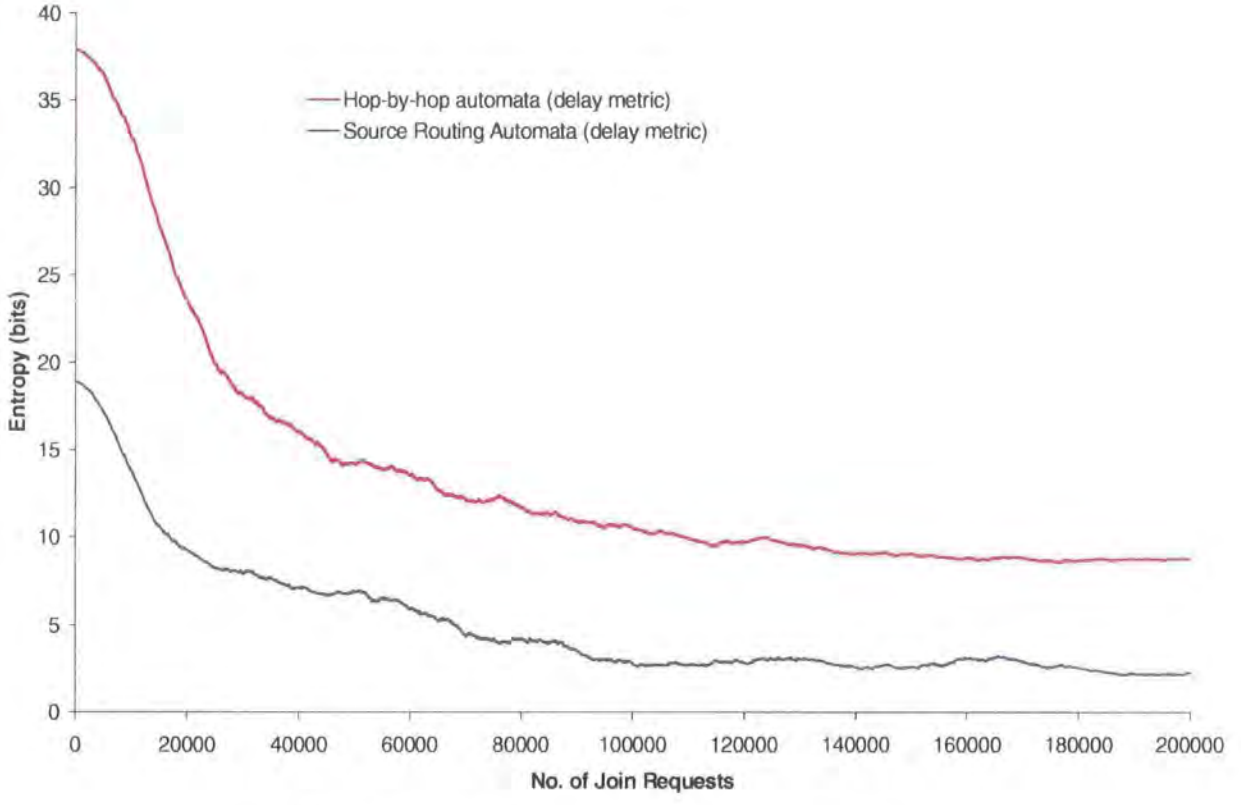


Figure 5.12 – Entropy plots, multiple source case.

5.6.3. Minimising Cost

The previous results show how automata minimise average packet delay at the receivers. Here, we are interested in minimising the total cost of the distribution tree where cost is defined as the total number of nodes in the tree. Since we are interested in cost rather than delay, we can set the source generation rates to 0 thus effectively transforming the simulator into a connection or session level simulator thereby speeding up simulations. Recall that automata learn here by attempting to minimise the number of hops travelled to join the tree, thereby minimising the number of nodes in the distribution tree. For this work, we have assumed hop-by-hop automata as described above. Initially, we have set up a single multicast group in the 30 node network consisting of the source (and single core) at node 24 and potential receivers at nodes 0, 1, 3, 4, 5, 6, 10, 11, 13, 14, 15, 17. On/off join times are initially set to 1/0.1 minutes respectively. We have monitored the number of hops required to join the tree in addition to the total number of members (excluding the source) belonging to the tree which we define as the cost of the tree. In addition to using automata to set up the dynamic trees, we have simulated shortest path trees where receivers route on the shortest path to the source and a random approach where the learning rates of the automata are set to 0. For this particular source and set of receivers, if all receivers join the tree, the minimum cost tree as given by the KMB algorithm is 13. The minimum cost tree is shown in Figure 5.13 and calculated using the package available from [153].

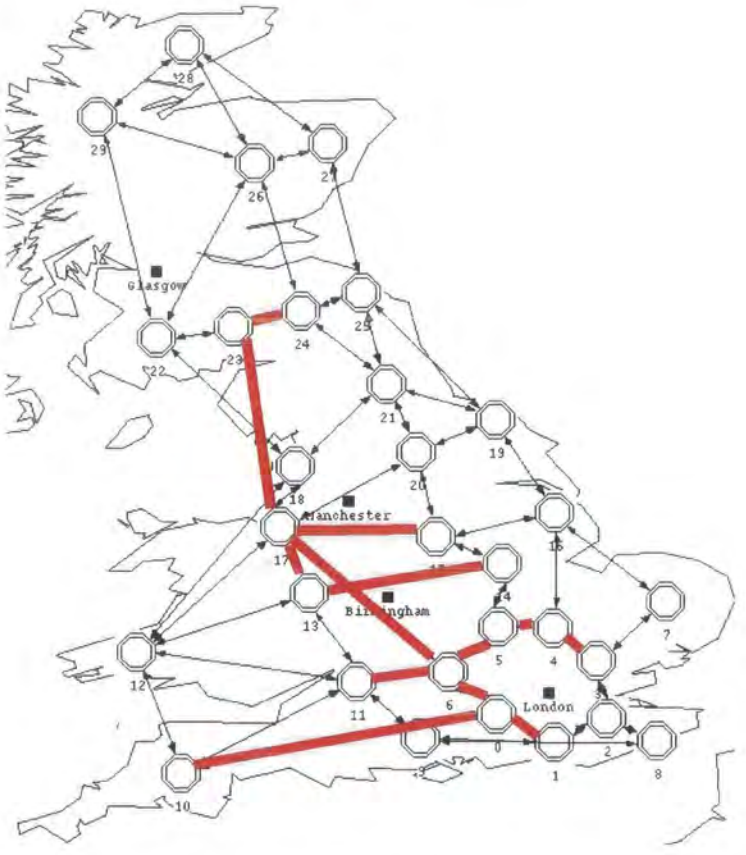


Figure 5.13 – Minimum Cost Tree.

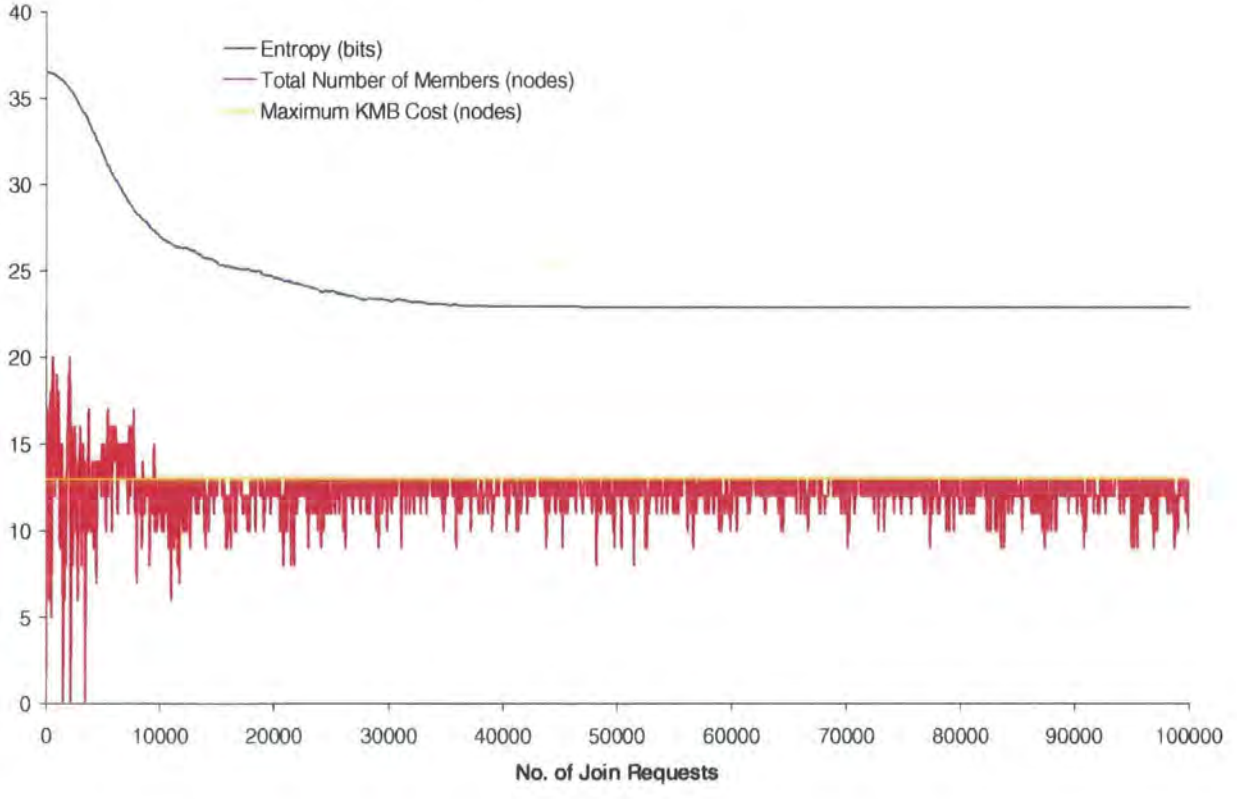


Figure 5.14 – Automata entropy and total cost sample paths.

In Figure 5.14, for the cost based automata and the multicast group as described above, we show sample paths of entropy and total number of tree members against number of join requests respectively. We see that the automata converge here in around 30,000 join requests, and learn to minimise the total number of members in the group which are dynamically joining and leaving the group. The maximum or worst case KMB cost (of 13) is also plotted and it can be seen that the automata converge such that the worst case cost (when all receivers join) of the dynamic trees is equal to that of the KMB algorithm. In Figure 5.15 and Figure 5.16, for automata, random and shortest path tree algorithms, we plot the average number of hops to join and average number of members against number of join requests respectively. All averages have been obtained using a moving average of window size of 1,000 connection requests. Figure 5.15 shows that the automata are clearly learning to minimise the number of hops to join the tree. Also, the variance in the average number of hops to join is considerably less for the automata indicating that there are less violent alterations in the distribution tree as receivers join and leave. Although the random routing algorithm produces a lower number of hops to join the tree than shortest path trees, we should remember that the traces only record the number of hops travelled for successful join requests. For this particular source and potential members, it turns out that 40% of join requests are not set up with random routing due to the formation of unicast routing loops.

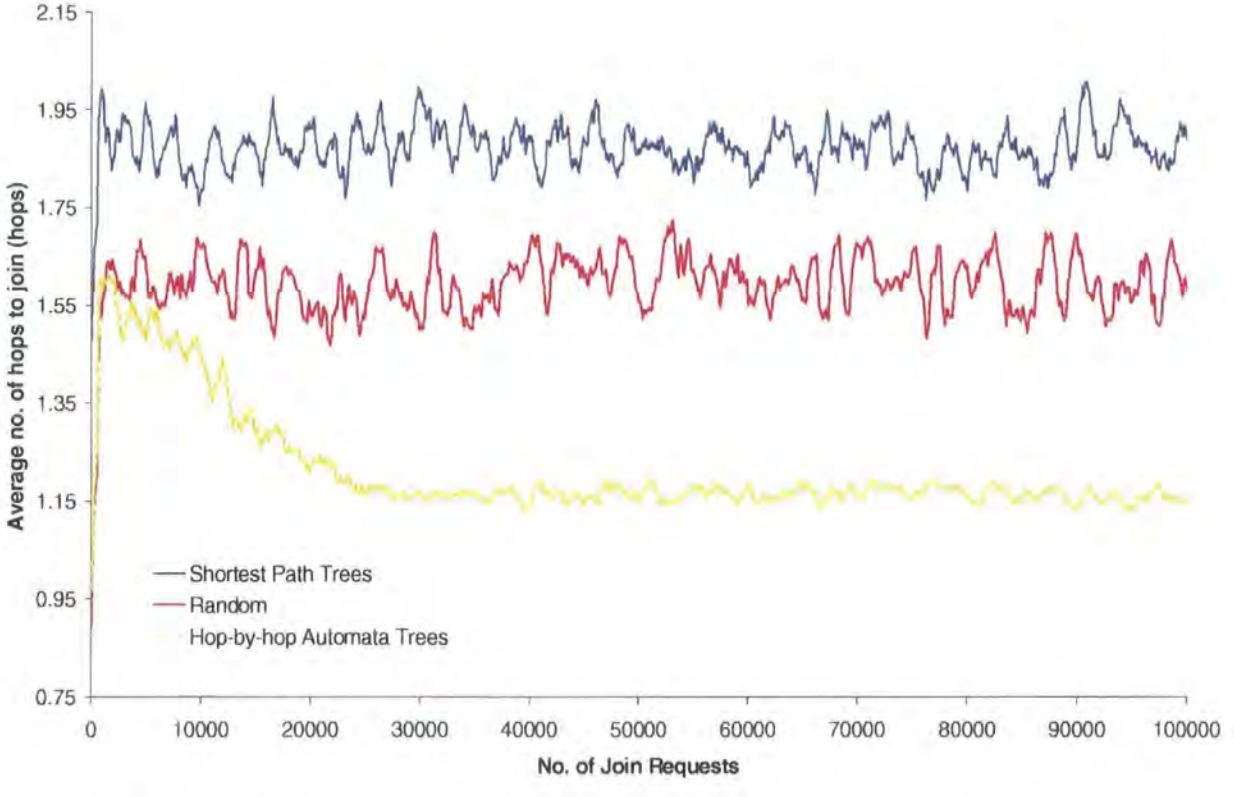


Figure 5.15 – Average number of hops to join the tree, all routing algorithms.

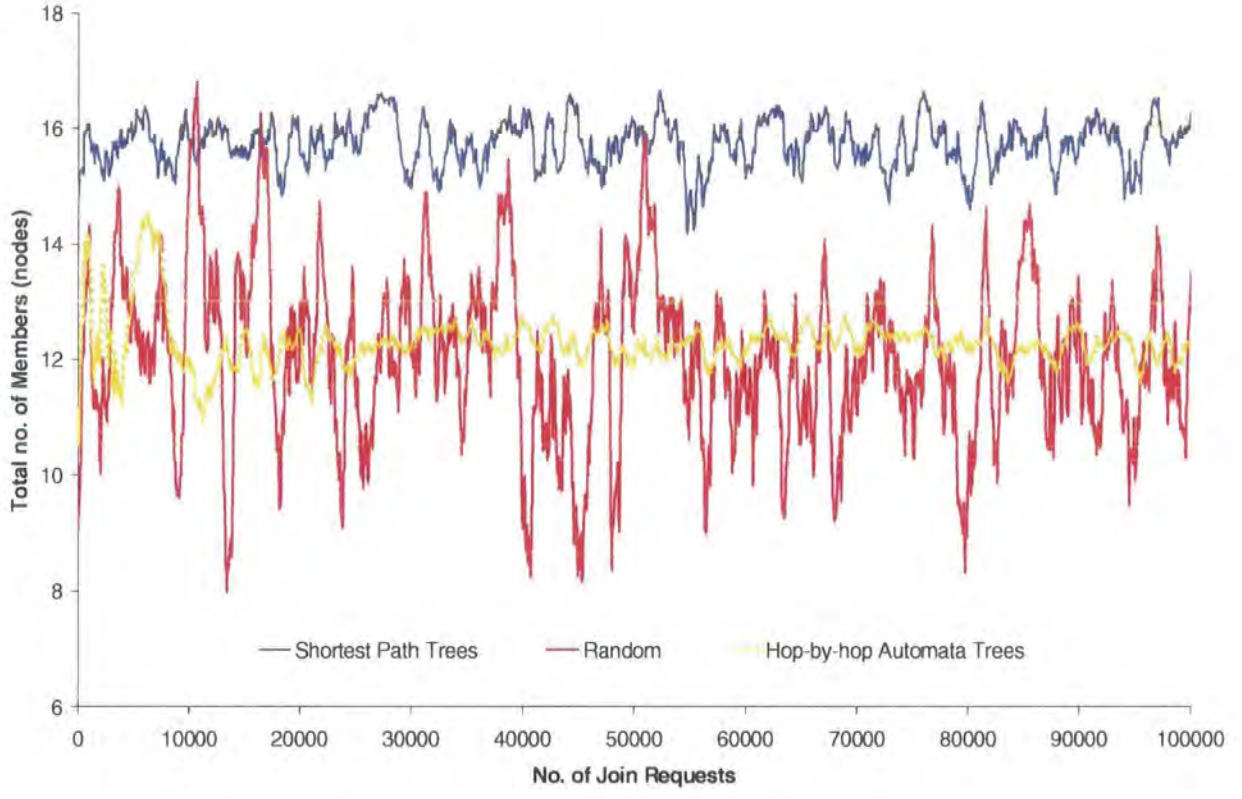
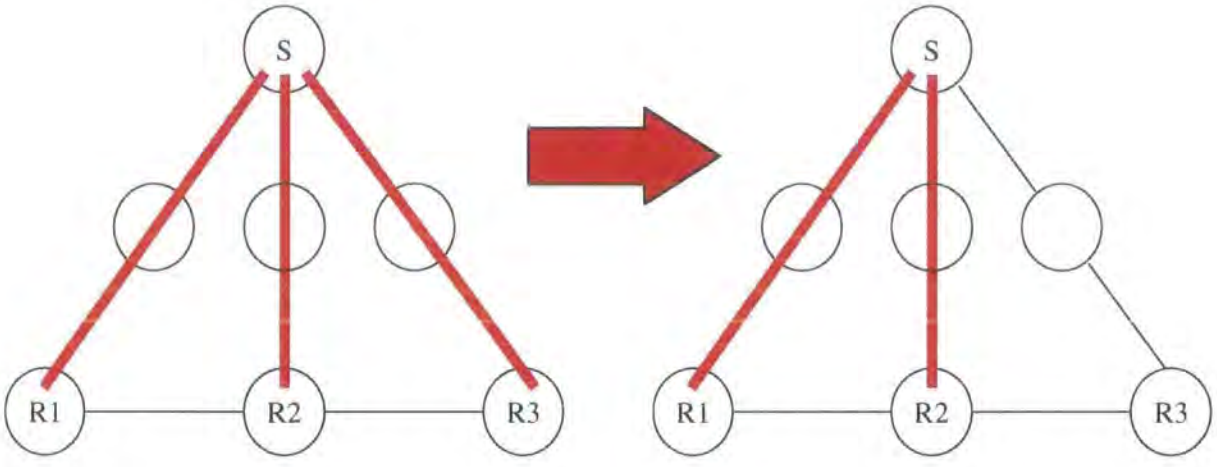
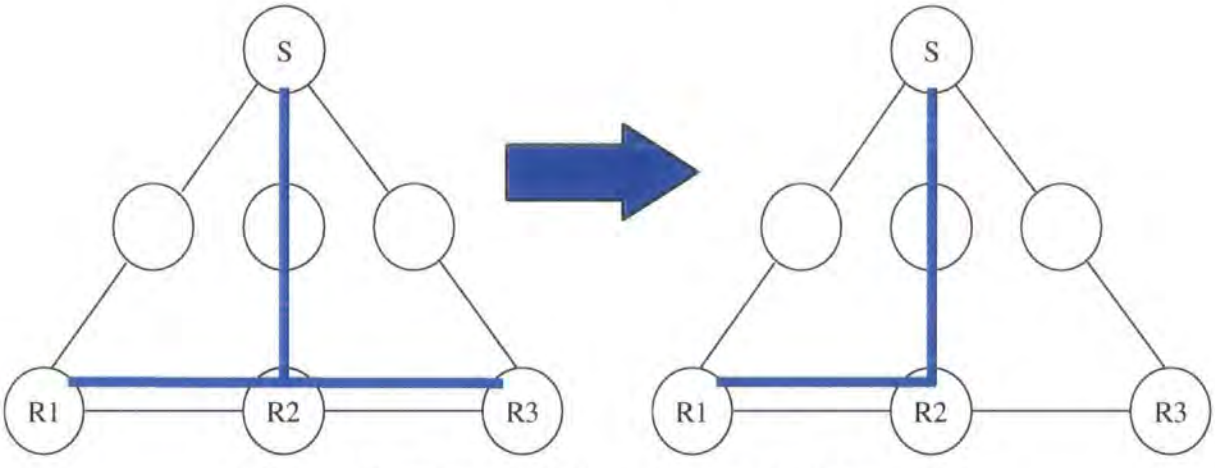


Figure 5.16 – Average number of total members, all routing algorithms.

Figure 5.16 confirms that the average cost of the automata trees is significantly lower in this instance than those produced by the shortest path trees algorithm, a difference in costs of around 30%. The random algorithm produces trees which vary considerably as indicated by the large variance in the number of members. The automata again are seen to produce trees with a lower variance than the shortest path trees. The reason for this is that the automata tend to align tree members to minimise cost such that when an external (leaf) node leaves the group, the majority of the other nodes within the tree remain in the group. Figure 5.17 shows the principle for a simple 8 node network, consisting of a source node S and three potential receiver nodes R1 through R3. In part (a), we show the shortest path tree before and after receiver R3 leaves the tree. In part (b), we show the same for the minimum cost tree. We see that the variation in cost due to receiver R3 leaving the group is less in the latter case explaining the lower variance in total members observed for the automata traces.



Shortest Path Tree, cost before = 6, after = 4, change = 33%.



Minimum Cost Tree, cost before = 4, after = 3, change = 25%.

Figure 5.17 – Effect of changing membership, shortest path and minimum cost trees.

For the above experiments, we have used a receiver on/off time of 1/0.1 minutes respectively. Here, we examine the effect of changing this parameter on the ability of the automata to learn the minimum cost trees. If the average on time is greater than the off time, there is a higher probability that a given node will be a member of the tree. The ratio of on time to off time therefore effects how many nodes will be members of the tree in the steady state. If this ratio is low, implying large off times relative to on times, join requests are less likely to meet members of the tree on their way to the source, such that it becomes difficult to construct low cost trees since there are very few tree members in the steady state and therefore less potential for sharing. In Figure 5.18 and Figure 5.19, we plot traces of average number of hops to join and total number of members for automata and shortest path trees for on/off times of 1/0.1 and 1/1 minutes respectively.

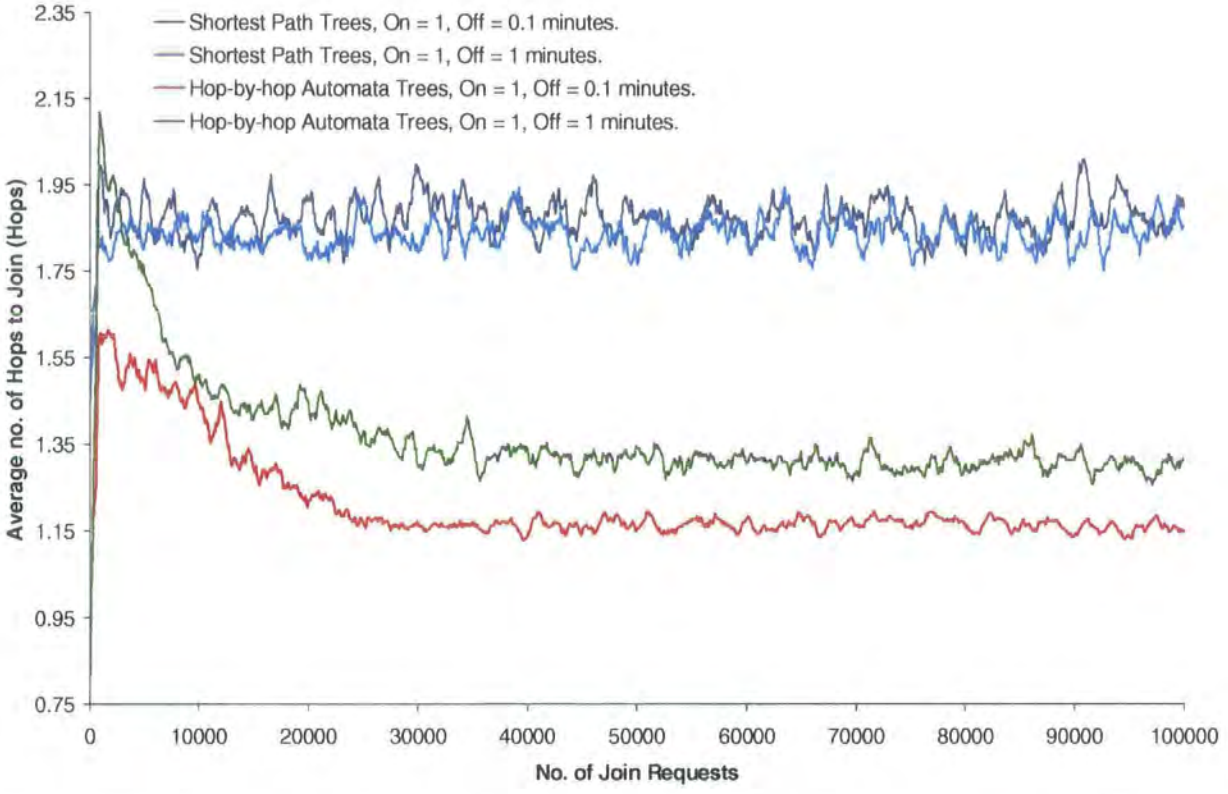


Figure 5.18 – Average no. of hops to join, shortest path and automata trees, various on/off times.

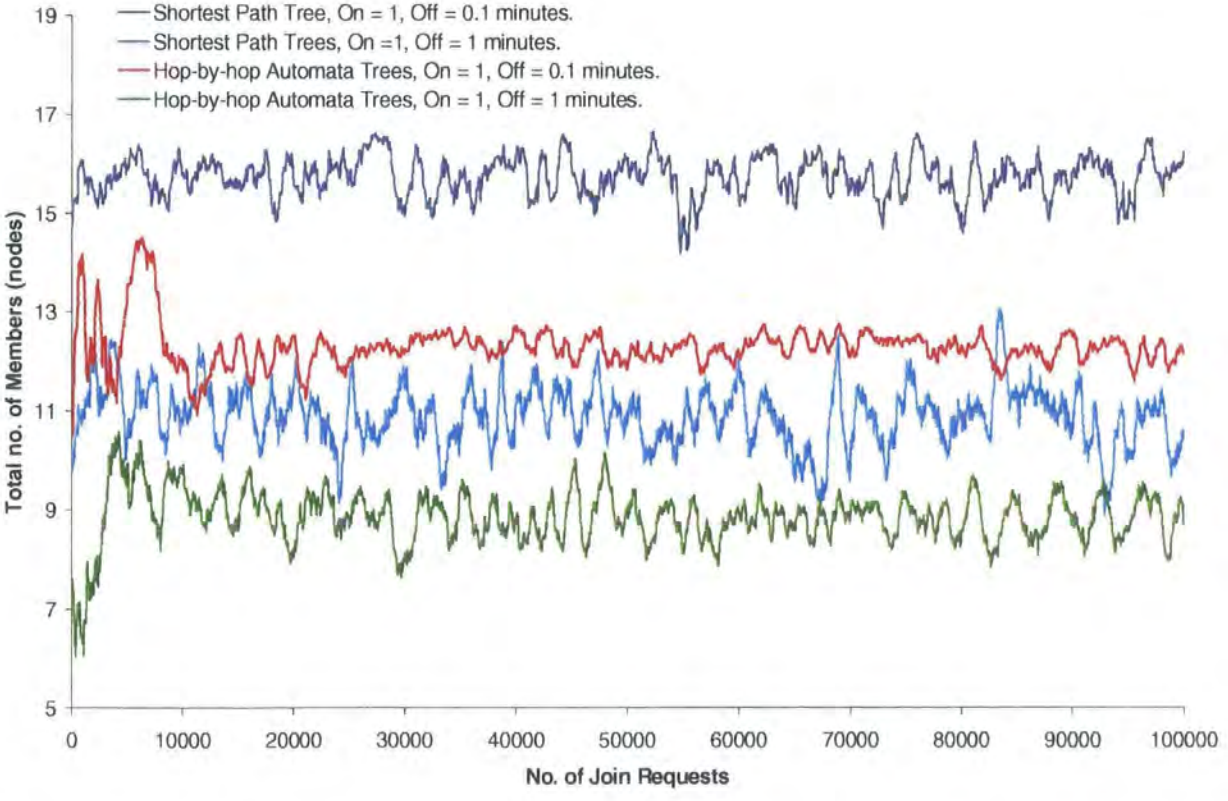


Figure 5.19 – Average number of members, automata and shortest path trees, various on/off times.

We see from Figure 5.18 that the automata are taking slightly more hops to join on average for the on/off time of 1/1 minutes as the number of members in the steady state has decreased. Also, the total number of member nodes in Figure 5.19 for the automata trees has decreased relative to the shortest path trees implying that is harder for the automata to learn of the minimum cost tree for low on/off time ratios as the probability of meeting an existing member is lower. With lower on/off ratios, there are less members in the steady state and therefore less sharing potential, so that any Steiner heuristic will perform less well relative to shortest path trees.

For the final cost based experiments, we have performed simulations for a single group where a single source (and core) is chosen at random and a certain number of potential receivers are also chosen at random. After a convergence period, we have measured the steady state number of nodes (i.e. cost) in the distribution tree. Finally, we let all potential receiver nodes join the tree at once, in a random order, and measure the total (static) cost of the tree. We have repeated this procedure for receiver on/off times of 1/0.1 and 1/1 minutes respectively. In Figure 5.20, we plot the steady state cost for on/off times of 1/1 and 1/0.1 minutes for shortest path and hop-by-hop automata based trees respectively. 90% confidence intervals are shown. The static costs produced are invariant to the precise on/off times used so in Figure 5.21, we plot shortest path and automata static costs for the on/off case of 1/1 minutes. We also plot the average static costs of the trees produced by the KMB algorithm. The code for the KMB algorithm was an augmented version of that made available by [154]. Confidence intervals are removed for clarity.

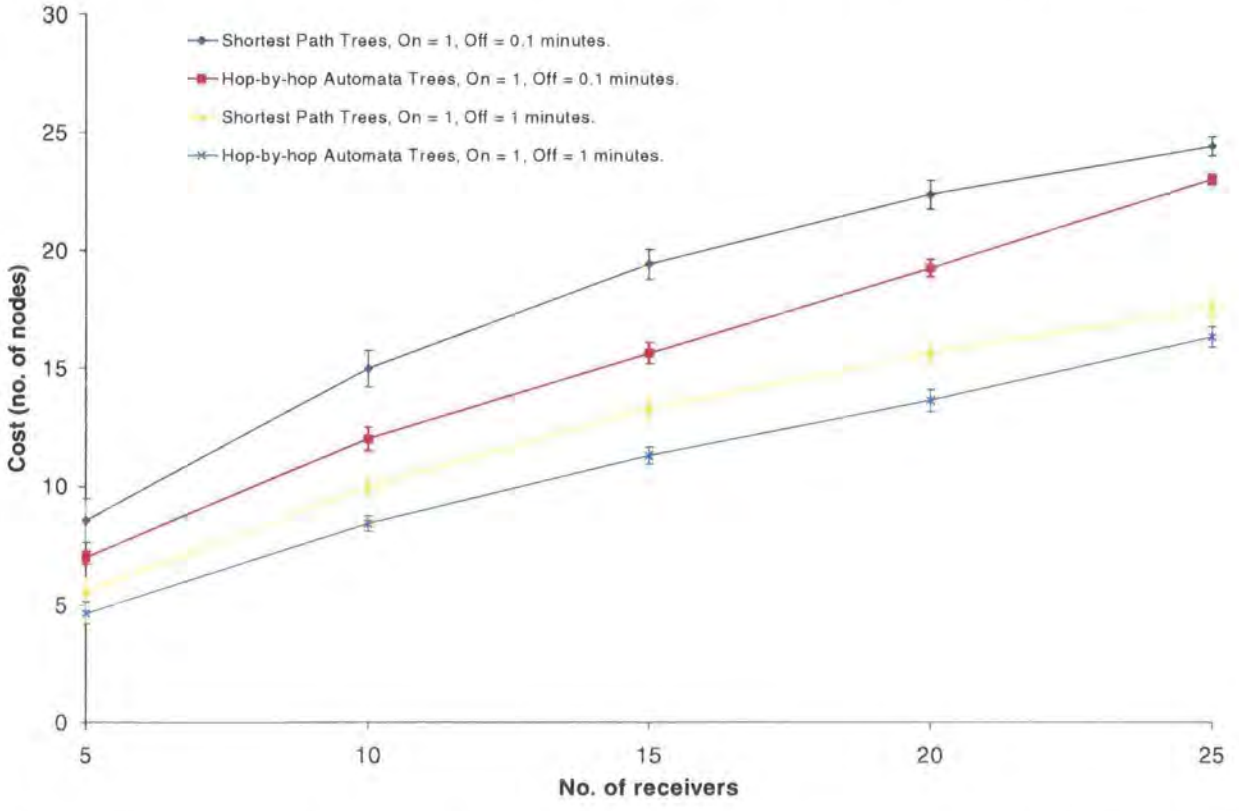


Figure 5.20 – Steady state dynamic cost, Shortest Path and Automata based trees, various on/off times.

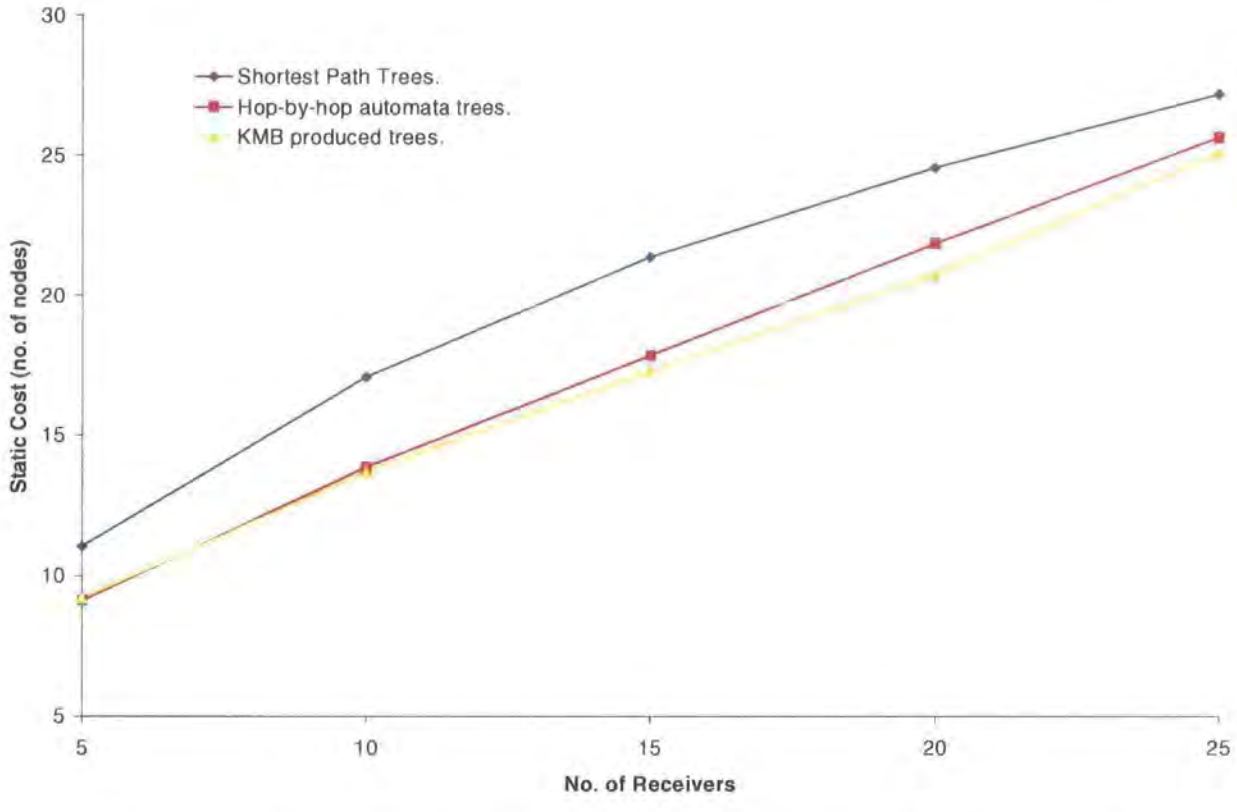


Figure 5.21 – Total static cost, Shortest Path, Automata and KMB based trees.

We see from Figure 5.20 that the automata produce lower steady state cost trees than the shortest path techniques which correlates with the transient results discussed earlier. Also, the relative difference in cost between shortest path and automata based trees is greater for average on/off times of 1/0.1 minutes as more nodes are members in the steady state, therefore improving the potential for the sharing of resources. Looking at the static costs in Figure 5.21, we see that the automata produce trees with costs extremely close to those produced by the KMB algorithm which is known itself to produce tree costs within a few percent of the minimum Steiner trees [151]. In effect therefore, the distributed set of learning automata to minimise tree cost represent an efficient minimum Steiner tree heuristic suitable for dynamic multicast environments.

A slight variant of the Minimum Steiner Tree (MST) problem is the 'Constrained Steiner Tree' (CST) problem. Here, given a source node and a set of potential receiver nodes, the aim is to minimise the cost of the tree joining all the nodes together whilst ensuring that the (propagation) delay from each receiver to the source is bounded by some value Δ. This problem is important since receivers will likely require a bound on the propagation delay for real-time service. Studies within the literature have generally focused on the static CST problem where the receivers are not joining or leaving the group (see [113] for overview, and also Chapter 6). Here, we study the use of learning automata for growing delay constrained trees in dynamic environments. To enable this, the simulation model is augmented so that a 'connect fail' signal is sent downstream whenever a connection set-up attempt fails the requested delay

bound. In this way, the LRI automata only reward those routes that meet the delay bound. Thus, a connection set-up request can be failed from either a potential routing loop or a delay bound failure. Delay bounds can be specified as a maximum value (i.e. Δ hops) or as a certain number of hops greater than the shortest path length for the particular receiver-source pair. Here, we adopt the latter approach, which assumes that all receivers have knowledge of the shortest path lengths to the source. Since the set-up packets record the number of hops travelled to join the tree, each node will know its distance from the source when it becomes a member (i.e. receives a join acknowledgement). When a connection set-up request arrives, we extract the number of hops travelled so far, together with the required delay bound. We add the number of hops travelled so far to the node's distance from the source and compare this with the required delay bound. We have run simulations as for the previously described experiment, with receiver on/off times of 1/0.1 minutes. We monitor the steady state (dynamic cost) and the static cost when all potential receivers join simultaneously. We compare the unconstrained automata, shortest path trees and the delay constrained automata trees with constraints of 0 and 2 hops greater than the shortest path lengths. We plot the steady state dynamic costs in Figure 5.22, and the static costs in Figure 5.23.

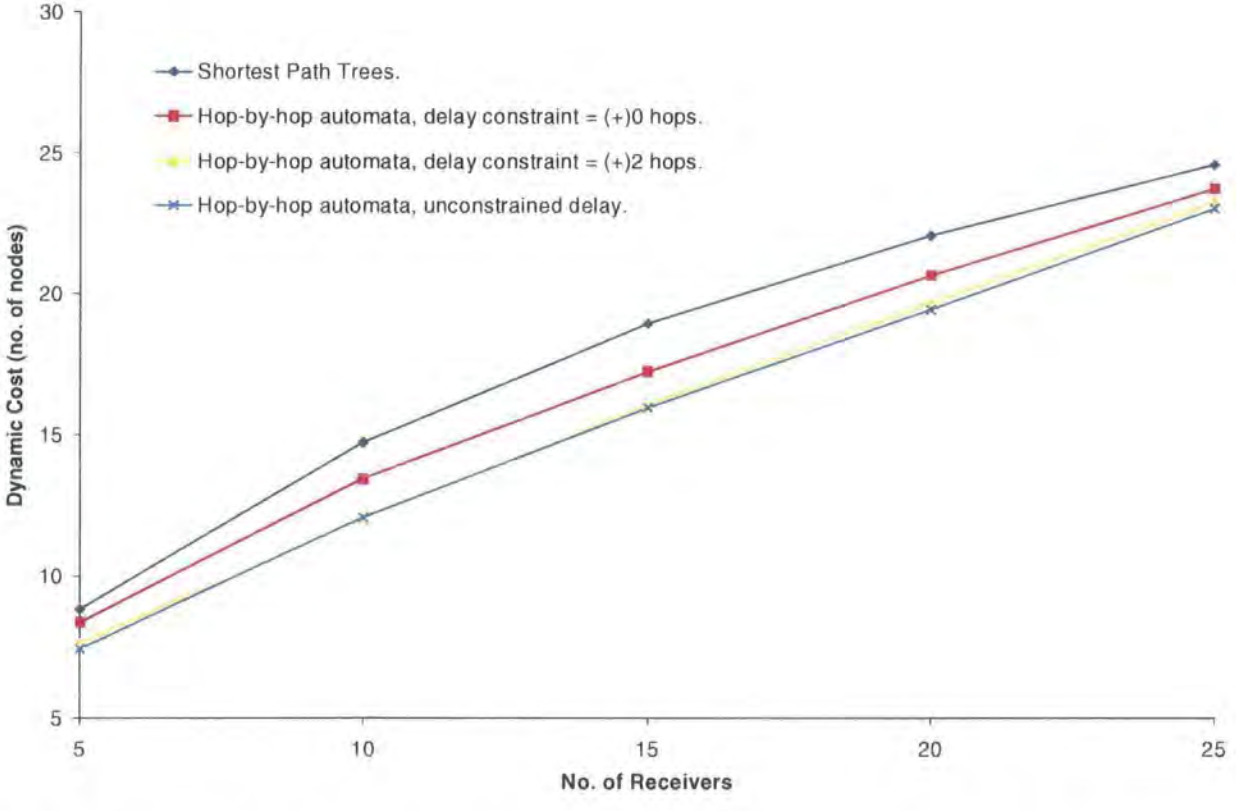


Figure 5.22 – Steady state dynamic cost, delay constrained automata, on/off times are 1/0.1 mins.

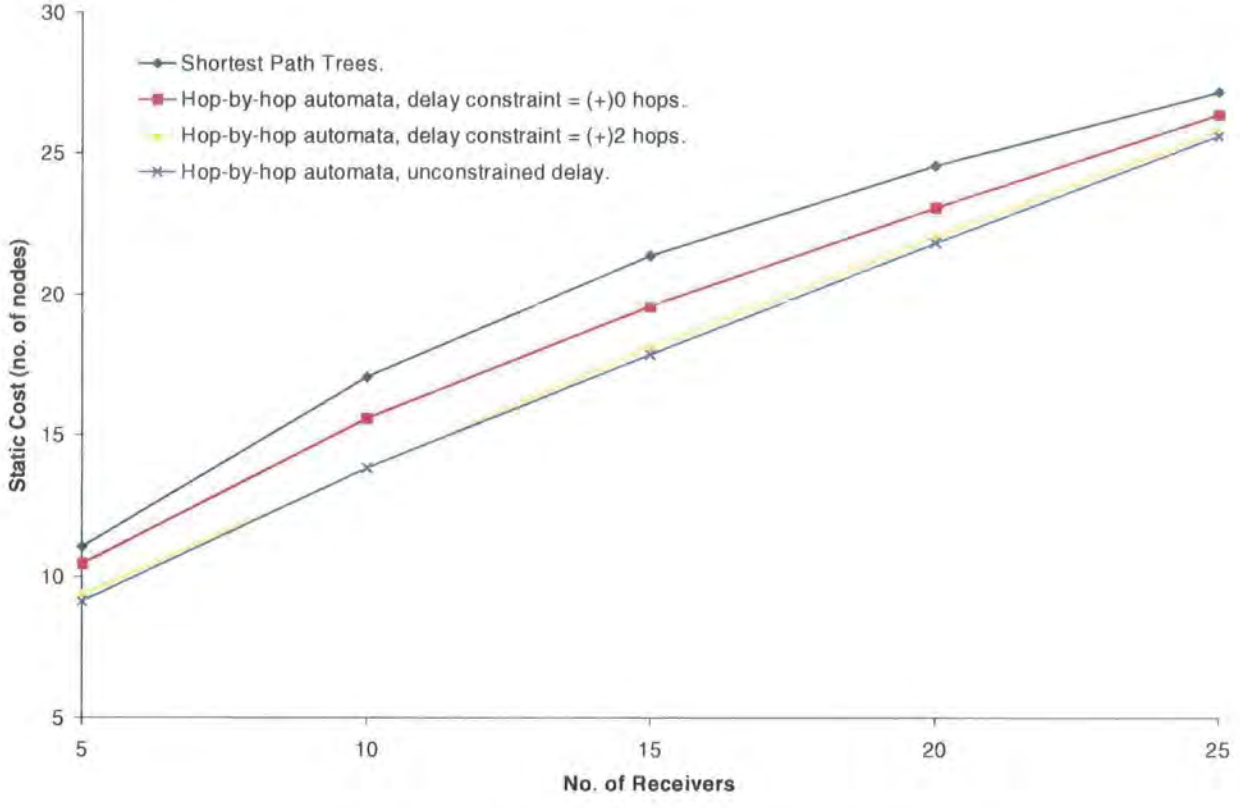


Figure 5.23 – Total static cost, delay constrained automata.

In Figure 5.22 and Figure 5.23, it can be seen that the automata with the tighter delay constraints produce higher cost trees in the dynamic and static cases respectively. As described at the start of the chapter, there is a fundamental trade-off between the cost and the (average or maximum) delay of a multicast tree. Steiner tree heuristics require some receiver-source pairs to take slightly longer paths to maximise sharing potential. Thus, constraining the length of the receiver-source paths places a limit on the amount of sharing that may take place in the tree and therefore, the resulting cost. Even the automata that are constrained to choose from all shortest paths (+0 hop constraint) improve costs over a shortest path tree however.

5.7. Summary

In this chapter, we have shown how learning automata may be used to construct multicast forwarding trees in dynamic best-effort environments. It has been shown that automata have the ability to grow the trees to minimise some performance index, average received packet delay and total tree cost (with or without delay constraints) having been considered. The primary motivation behind the application of learning automata has been to improve the delay/cost characteristics of shared trees whilst retaining their scalability advantages, particularly in an integrated traffic environment where there could be high variance in the 'left over' resource available to best-effort traffic. For the delay based trees, automata have been shown to minimise the overall delay of the shared tree for a single group in the single and

multiple source case. For the minimisation of tree cost, automata have been shown to produce tree costs comparable to those produced by the KMB algorithm for the 30 node network considered. In both cases, a receiver oriented model has been considered where receivers do not require knowledge of the location of other group members or link costs/delays for the network. In the worst case, automata only have knowledge of their directly connected neighbours and at most have knowledge of the network topology therefore representing a possible solution for the inter-domain mutlicasting problem.

The work in this chapter has concentrated on the data multicasting problem where no resources are reserved in the network to guarantee a specific end-to-end delay bound. There may be a place in future networks for constructing multicast trees which do guarantee throughput and/or delay to the receivers (i.e. provide a certain QoS). A typical application might be an interactive group lecture where the real-time constraints are such that users wish to reserve resources within the network. In the next chapter, we investigate the feasibility of applying automata to the QoS Multicast Routing problem.

Chapter 6

Learning Algorithms for Quality-of-Service (QoS) Multicast Routing

6.1. Introduction

In the previous chapter, we saw how learning algorithms could be used to grow shared multicast trees which minimise some performance index such as average delay at the receivers whilst maintaining the attractive scaling properties of a shared tree approach. In this chapter, we examine the applicability of learning algorithms for growing multicast trees that meet QoS demands such as bandwidth and propagation delay. This is an important issue for envisaged applications such as interactive video conferencing where we would like provably low delays between the participants. The challenge is to construct algorithms which may grow QoS-bounded multicast trees whilst retaining the low overheads necessary for implementation in real communication networks. Firstly, we introduce the problem of QoS-based multicasting, providing a brief review of some proposed solutions within the literature. We then go on to describe how learning algorithms may be applied to the problem and document the benefits of doing so. The simulation model is then described, results are presented for per-source multicast trees and shared trees. The final section examines the routing of unicast and multicast traffics contained on a single network, to examine whether one set of automata can route both effectively. Overall conclusions are then drawn.

6.2. Background

The problem of routing multicast connections to meet QoS constraints is starting to receive considerable attention in the literature (see [113], [141], [143], [151], [155], [156], [157], [158]). The majority of this work has focused on creating static trees which minimise overall costs, possibly subject to a delay constraint. The problem is usually formulated as a source node sending to a fixed set of receiver nodes in the network, so we are concerned with source routed trees (i.e. point-to-multipoint). The aim is to minimise the total cost of the multicast tree joining all these points, where the cost is defined as the total number of links or nodes for example. This problem is known as the 'Minimum Steiner Tree' (MST) problem (as observed in the previous chapter), and is well known to be NP-complete [140]. For this reason, a number of heuristics have been proposed that can produce trees with average costs of only a few

percent more than the minimal cost tree (see [140] for a summary). A more recent variant of the MST problem has been defined, where the aim is to minimise the overall cost of the (static) tree whilst ensuring that the delay from the source to each receiver in the tree is bounded. This problem is known as the 'Constrained Steiner Tree' (CST) problem [157] and has also been shown to be NP-complete [157]. A good summary of the heuristics proposed for the CST problem is contained in [113, 141]. The application of Steiner Tree (MST or CST) heuristics in real networks is rather unrealistic for a number of reasons. Firstly, the majority of the proposed heuristics require global knowledge of the cost and delay associated with each link in the network, where the cost is commonly set to be the utilised bandwidth of the link, which could be changing frequently. As such, they are usually posed as a centralised computation and are not therefore suited to the distributed nature of communication networks. Secondly, it is unlikely that multicast group membership will be static, and we would expect nodes/receivers to join and leave the multicast groups dynamically. Many of the Steiner heuristics are computationally intensive, some having a time complexity of between $O(N^3)$ and $O(N^4)$ (see [113]) where N is the number of nodes in the network, and there could be considerable computational effort required if we have to compute the optimal tree configuration each time a node joins or leaves the multicast group. Furthermore, we would most likely need to tear down the existing tree in order to set up the new 'optimal' tree which could mean significant disturbance to those nodes already participating in the multicast session. Some packets may be lost or delayed and may arrive out of order at the destination. A slightly less drastic approach is to partially rearrange the tree, perhaps through local modifications, although this would also lead to disruption to some extent. The approach leading to least disruption is to make only incremental changes to the multicast tree as nodes join and leave. This means that the cost of the tree will fluctuate with the random behaviour of receivers joining and leaving, although a good algorithm should produce a low cost on average. We refer to this final approach as the 'dynamic multicast' problem. Some papers have examined the dynamic multicast problem for a single multicast group in isolation [143, 151, 156]. Here, the aim has been to quantify the inefficiency of simple approaches to multicast tree construction such as shortest path trees as compared to more complicated Steiner heuristics. The shortest path algorithm is the easiest to implement for multicast routing purposes and has been used extensively for unicast routing, and may be formulated as a centralised or distributed computation (e.g. OSPF [68], RIP [92]). The shortest path algorithms only minimise resource consumption as a side effect, when the shortest paths converge, and make no effort to minimise the cost of the overall tree. For this reason, shortest path algorithms have been referred to as 'naïve' routing [143], since they require no explicit knowledge to construct the tree. Studies have shown (see [113], [143], [145], [151]), that simple shortest path trees produce trees with costs of around only 50% greater than the optimum for node connectivity degrees at Internet levels (i.e. around 3-5). This questions the use of complicated Steiner Tree heuristics altogether. There has to date been very little work examining the more complex case of the dynamic multicast problem for multiple groups, in an environment where there

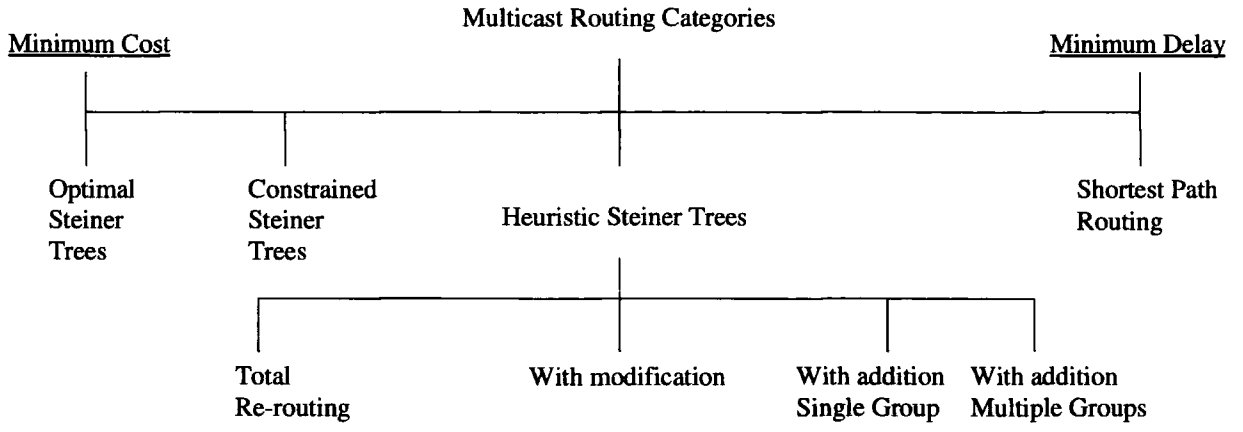


Figure 6.1 – Approaches to Multicast Tree Construction

is contention for resources between the different groups. This is the problem we tackle in this chapter. In Figure 6.1, we summarise the possible approaches to multicast tree construction. This Figure is a slightly altered version of that presented in [143]. It would appear that a source routed shortest path tree has the advantage over CST algorithms in terms of the trade-offs between complexity and performance. However, since each node only maintains a single shortest path to the source of the group, a node cannot choose an alternate path if the primary path is congested due to usage by other multicast groups or other services (e.g. unicast). We know that if the variance of resource availability is high, an adaptive control strategy can improve performance through load splitting. It would possibly prove beneficial in these cases if nodes could choose between a number of paths to the source to enable load balancing to take place and increase overall throughputs. When we consider the most complex case of dynamic multicasting for multiple groups in a resource limited environment, we want to create low cost trees whilst load balancing at the same time. From the previous discussion, we can summarise the required attributes necessary for an algorithm to be applied to the dynamic, multiple group multicasting problem. The algorithm should:

1) Maximise the number of concurrent multicast sessions through minimising the cost of the multicast trees in addition to load balancing between the trees.
2) Bound the (propagation) delay of the source of any group to the receivers of that group.
3) Work in an incremental fashion.
4) Enable a distributed computation.
5) Utilise minimum network state information i.e. without necessarily knowing the cost (utilisation) of the links in the network.

We intend to show how learning algorithms meet all of the above points, whilst providing a close to optimal solution.

6.3. Simulation Model

For this work, we assume a receiver-oriented model, which has the advantage over source-oriented models in that it can deal with heterogeneous receiver requirements [93]. The emphasis is placed on receivers to find out what QoS the multicast group (i.e. the source) may support and is an example of control propagating to the edge of the network. One proposed source-oriented mode is QOSPF [159]. With this model, in order for the source to send messages to the receivers, it must have knowledge of which receivers are in the multicast group and uses receiver location broadcasts to obtain this information, leading to a significant receiver discovery overhead. In addition, it incurs a considerable path computation overhead due to the need to keep track of existing flow paths. In order for the path computation to scale to large dynamic networks, a receiver-oriented model would seem to have the advantage. According to [93], under this model:

1) Sender traffic advertisements are multicast over a best-effort tree which can be different from the QoS accommodating tree for sender data.
2) Receiver discovery overheads are minimised by utilising a scaleable IDMR (inter-domain multicast routing) scheme (e.g. PIM [138], CBT [152] etc..), to multicast sender traffic characterisation.
3) Each receiver independently computes a QoS-accommodating path from the source, based on the receiver reservation. This path can be computed based on unicast routing information only (e.g. shortest path routing), or with additional multicast flow-specific state information. In any case, multicast path computation is broken up into multiple, concurrent unicast path computations.
4) Nodes processing unicast reserve messages from receivers aggregate resource reservations from multiple receivers.

In this chapter, we are concerned primarily with steps 3 and 4 above. That is, we assume that receivers know the address of the source and the QoS (bandwidth) it may support, this information possibly having been distributed on a scalable best-effort multicast tree as described above. For the unicast QoS-accommodating path computation in step 3, we believe that the computation should be extremely lightweight (as for QoS-unicast routing) to enable practical implementation. Specifically, the automata described here do not require multicast flow-specific state information, only that nodes receive a simple binary feedback signal indicating the success or failure of a connection set-up attempt to the multicast group.

The basic process for a node/receiver joining a multicast group is as follows. The receiver wishing to join a particular group generates a join-request for that group which is forwarded to the source of the group by the routing algorithm. Upon arrival at a node, a number of checks are made before accepting and forwarding the connection set-up request. These involve checking for sufficient free resources at the

node, checking that no routing loops will be formed and possibly checking that the set-up still meets any delay bounds specified by the receiver. Once the join-request successfully reaches the source or an existing member of the group, a join acknowledgement is sent back to the initial requesting receiver. If at any point, the set-up fails any of the above checks, a 'connect fail' signal is sent back to the requesting receiver. Thus, a receiver will always receive a positive or negative (binary) response. It is this response that provides the basis for the application of learning.

Each multicast group is represented by a different source in the network. Thus, for example, 5 multicast groups will be represented by 5 different sources in the network. In addition to the number of multicast groups, we can vary the number of potential receivers for each group. In the simulations that follow, we select a number of sources, chosen at random, to represent multicast groups and a certain number of potential receivers for each multicast group, also chosen at random. If a node is chosen as a potential receiver for a particular multicast group, it will generate join/prune requests for the source of the group according to an exponential on/off distribution in a similar manner to the dynamic mulicast experiments carried out in the previous chapter. An existing member of a multicast tree may only leave the group if it has no children. This is known as 'route-pinning' [94] and prevents any disruption of data flows to receivers due to membership changes in the multicast tree, which may be important for future real-time multicast services. All receivers are assumed to make equal reservations to the source of the group(s). Thus, we are simulating a homogeneous receiver requirement although we see no reason why automata could not be used for heterogeneous receiver requirements where the source sends at the maximum QoS and receivers may choose independent QoS levels. In the simulations that follow, each node can support a maximum of 10 reservations. In other words, any node can be a member of a maximum of 10 different multicast groups. Thus, by having more than 10 multicast groups in the network simultaneously, we have a probability that a join request will be blocked due to lack of resources.

6.4. Source routing automata

The aim here is to use learning automata to select between a number of pre-computed paths to the source of the group. Each receiver has a learning automaton for each multicast group (i.e. each source), the actions of which represent choosing one of a finite number of paths to the source of the multicast group. In this way, we hope that automata should provide load balancing in the network by utilising sensible use of alternate paths, thus increasing the chance that a join to the group can be accepted. We assume that for real-time service, applications at the receivers will require bandwidth and propagation delay guarantees. The alternate paths could possibly be derived from an existing unicast routing protocol. The method we use to select alternate paths is to first choose from the set of shortest paths, then shortest paths plus one etc.. until we have the required number of paths to the source of the group in question. The propagation delay to the source of the group can be bounded by choosing alternate paths of sufficient length. The idea is to construct the alternate paths from relatively static information such as topology or QoS capability of

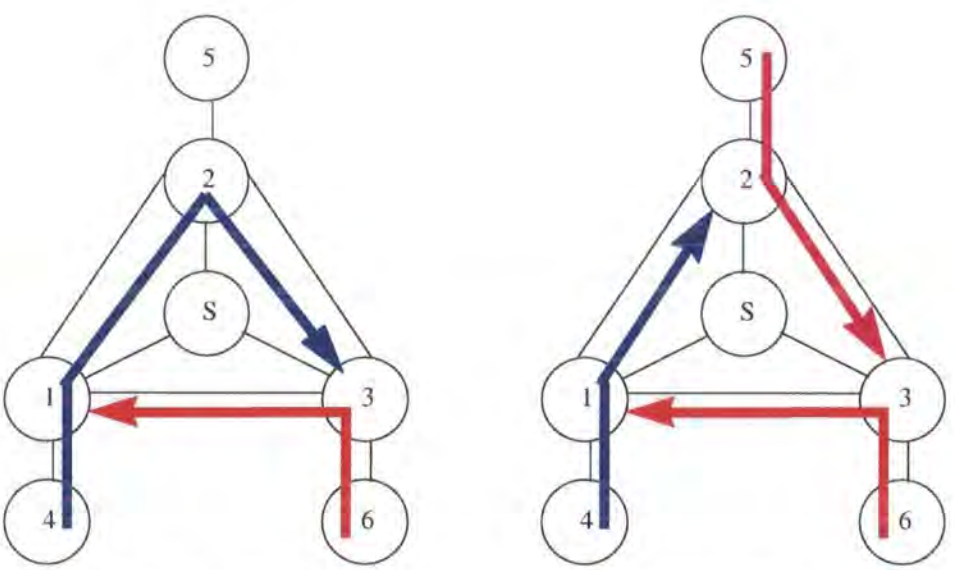
a particular node rather than dynamic flow state information (i.e. dynamic loading information).

To enable learning to take place, we assume that receivers use the explicit join and leave process for a particular multicast group as described previously. Thus, a receiver always gets a feedback signal telling it of the success or failure of its join request as described above. Based on this binary feedback, the learning automaton for the group in question updates the probability of using the chosen path for future join requests. A P-type automaton is used due to the binary nature of the feedback (Appendix A).

Join requests are source routed, so that each join packet generated by a receiver contains the complete path from the receiver to the source of the group. Source routing is a simple unicast loop prevention mechanism, although requires larger fields in the packets to store the complete address, and assumes that nodes have complete topological knowledge to calculate the source routes. Thus, on arrival at a non-member node, the join request is simply forwarded to the next node as dictated by the source route stored in the join request packet, providing that there are enough spare resources and there is no routing loop potential as described below.

6.4.1. Avoiding Routing Loops

When using alternate paths to construct multicast trees, routing loops may be formed due to different receivers choosing different paths. In Figure 6.2 taken from [94], we show how routing loops may be formed by two and three set-ups respectively.



a) Loop formed by two Set-ups b)Loop formed by three Set-ups

Figure 6.2 - Loops Formed by Set-up Messages

In Figure 6.2 (a), nodes 6 and 4 are using alternate paths to the source S, and it can be seen that if the set-up from 4 reaches 1 first and the set-up from 6 reaches 3 first, the set-ups may merge and form a routing loop, so that nodes 6 and 4 will be waiting for an acknowledgement (or a connect failure) indefinitely. This loop may be prevented if nodes 1 and 3 compare the route upstream with that proposed, to check for

potential routing loop formation. Figure 6.2 (b) shows a similar scenario where a routing loop is formed by three set-ups. Here, the set-up from 4 reaches 1 first, the set-up from 5 reaches 2 first and the set-up from 6 reaches 3 first. To prevent the formation of routing loops then, we require that each set-up match the upstream route already in place on the tree as adopted in [94]. Upon arrival at a node, if there is an outstanding join request, the node will check that the source route contained within the recent join request matches the route upstream. We know the route upstream since a join request that passes through the node in question contains the complete source route, which we record. If the routes do not match, a connect fail will be sent back to the previous node of the most recent join request. Thus, the three ways in which a join request may fail are from insufficient capacity, delay bound violation and from potential routing loop formation.

6.5. Hop-by-hop automata routing

In this application of learning automata, the automaton at each node (again, one per group) simply selects an output link rather than a complete source route. In this way, we do not require the node to have a complete picture of the network topology, only knowledge of its directly connected neighbours. In this approach, learning automata learn to select from all possible paths rather than being constrained to choose between a finite number of source routes. P-type automata are also adopted here and the chain of automata involved in a routing decision are updated based on the binary success/failure response. To prevent the formation of unicast routing loops, each join request packet stores the sequence of nodes selected by the distributed learning automata, to check that a set-up packet does not visit the same node twice. If a unicast routing loop does form, a connect fail signal is propagated down the path. Although this takes care of unicast routing loops, special measures must be taken to avoid the formation of multicast routing loops.

6.5.1. Avoiding routing loops

Like the alternate source routing technique described previously, we require that the alternate route chosen by a node match the route already in place on the tree upstream. However, unlike the source routing technique, there is no way of knowing *a priori* what route will be chosen upstream by the independent learning automata located at each node. Thus, the only way to prevent the formation of routing loops is to only allow one outstanding join request at any one time, thus preventing the merging of join requests. This will only prove a problem when there is a significant probability of more than one join request arriving at a node within close proximity. This should only be the case for extremely dynamic, large multicast environments where receivers are joining and leaving the group very frequently.

6.5.2. Meeting delay constraints

With the source routing approach, delay constraints were met by selecting alternate paths of sufficient length. With a hop-by-hop automata approach, meeting delay constraints is more complex. To achieve

this, we firstly require that each node know the shortest path length (and therefore propagation delay) to the sources of the multicast groups. This knowledge could probably be obtained from an existing unicast routing protocol, although on-line estimation can be used as described in Chapter 3. Since join request set-up packets record the route taken, we know the length of any paths set-up when a node becomes a member. Thus, any member of the multicast tree should know the current distance from itself to the source of the group. When a node wishes to join, we can therefore calculate the distance (in hops) to the source of the group, making sure it meets any delay constraint requested by the receiver. Nodes must have a knowledge of the shortest path length to the source so that they can request a feasible delay constraint in the first place. In this way, learning automata should learn to select paths which meet the delay bound to the source whilst minimising global blocking probability. An alternative way to bound delay is to have an upper bound on the path length between any receiver and the source, rather than bounding the path length to be within a certain number of hops of the shortest path length. This is the approach taken for most previous QoS multicasting work. This approach is also possible with learning automata, and removes the need for nodes to know the shortest path length to the source although can lead to overly conservative or lax bounds for certain receiver-source node pairs. We adopt this approach when considering the routing of QoS-based shared trees in section 6.7.

6.6. Results

To investigate the viability of the learning automata approach to the construction of real-time multicast trees, simulations have been carried out on the 30-node network shown in Figure 3.3 in Chapter 3. Firstly, we look at some steady state results showing blocking probability versus number of receivers per multicast group for varying numbers of multicast groups. For these simulations, each node may support a maximum of 10 different multicast sessions. i.e. a node may only be a member of 10 different multicast groups at any one time. We compare 4 different receiver-oriented multicast routing techniques, single shortest path, source routing automata, hop-by-hop automata and random (hop-by-hop) routing. In addition, we have simulated shortest path and learning automata unicast routing where paths cannot be merged. This enables us to compare the relative savings of a multicast approach over a unicast approach. One of the goals of the simulations was to investigate the viability of alternate path multicast routing. Work on adaptive unicast routing (see [93] for overview), has shown that adaptive routing can lower blocking probabilities and increase network utilisation through load balancing. Thus, although calls may take longer paths and consume more resources, the overall effect is to spread the load more effectively and allow more calls into the network. With adaptive multicast routing, the situation is more complex since we must consider the effect a chosen route will have on the sharing of resources. The aim of multicast routing after all is to maximise the sharing of resources through the use of a practical algorithm. If we choose a longer alternate path in the multicast context, we may or may not utilise more resources in joining a tree, depending on the particular members of the multicast groups at that instant. Recall that

Steiner trees usually have some source-destination paths take slightly longer routes to maximise sharing, which is why Steiner trees also usually have higher average delays. The aim then is to take those paths which lead to increased sharing, therefore leaving more resources for future connection requests, whilst load balancing at the same time. In general, it may be better to form a slightly higher cost tree which spreads the load rather than form the minimal cost tree according to the layout of the members at that instant.

Figure 6.3 shows blocking probability against number of receivers/group for the case when there are 15 multicast groups, for the 4 different routing techniques. For all learning automata algorithms reported subsequently, LRI reinforcement is adopted with a learning rate of 0.02. All source routing automata store two paths to the source of the group. For the hop-by-hop automata, the paths chosen are constrained to be two hops greater than the shortest path length (i.e. + 2 hops). For the unicast learning automata, requests could travel a maximum of 13 hops. 90% confidence intervals are shown. Receiver on/off times are 1 minute and 0.1 minutes respectively.

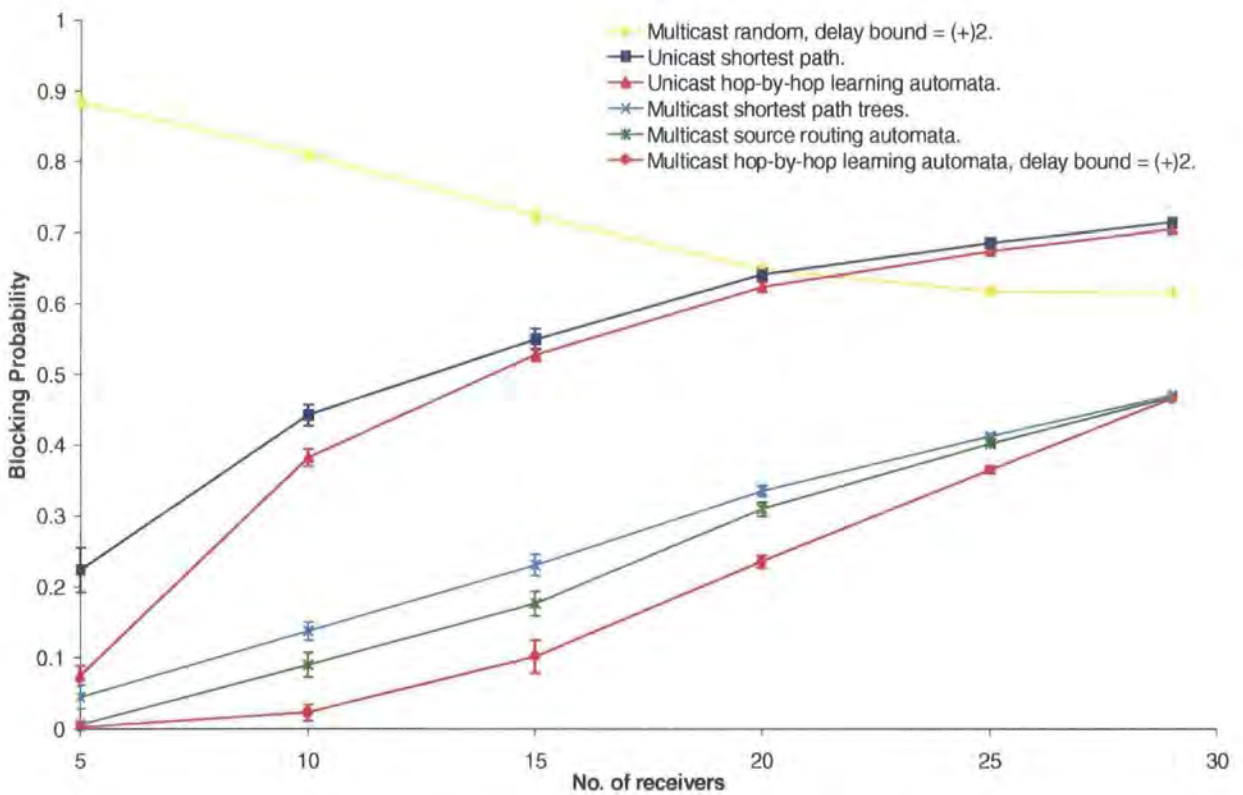


Figure 6.3 - Blocking probabilities, 15 groups.

It can be seen from the above plot that in terms of blocking probability, the source routing automata improve marginally over the shortest path approach and that the hop-by-hop automata improve over source routing automata by a similar degree. In a similar manner to the unicast QoS-routing results (see Chapter 3), the maximum difference between shortest path and learning based routing occurs for mid-

levels of contention. Here, this corresponds to multicast groups with around 10-20 members. All multicast routing approaches reduce blocking over unicast routing, although the difference is quite low for small multicast groups where there is little potential for sharing of resources. Interestingly, the random routing method has decreasing blocking probability for increasing number of potential receivers in the network. This is because the majority of failures are due to delay bound failure and as the number of receivers increases, there will be a higher probability of meeting an existing group member within the required delay bound (+2 hops in this case). In Figure 6.4 and Figure 6.5, we repeat the above experiment for 20 and 25 concurrent multicast groups in the network whilst discarding the random routing trace. Again, it can be seen that the hop-by-hop automata construction of real-time bounded multicast trees leads to considerable improvements in blocking probabilities, for mid-level blocking probabilities, although the curves are more compressed as the general level of blocking has increased in the network. For high loadings, it can be seen that shortest path routing has the edge, a similar result to the unicast counterpart, where longer alternate paths interfere with directly routed shortest paths. In Figure 6.6, we depict three plots of hop-by-hop learning automata for the 20 multicast group case. We investigate the effect of varying the hop count bound or delay requirement of connection requests. The three plots shown represent a hop count bound of 0, 2, and 5 hops greater than the shortest path length respectively. The performance of the hop-by-hop automata scheme is found to increase for less stringent hop-count bounds at low to mid loadings as automata can make use of longer paths to maximise sharing and load balancing. However, the automata schemes with tighter bounds perform better at high loadings since they use less resources via alternate paths. In Figure 6.7, we show the average number of hops a join request must travel to join a tree against the number of receivers per group for the 25 multicast group case. This gives us some idea of the degree of sharing and load balancing that is taking place. 90% confidence intervals are shown.

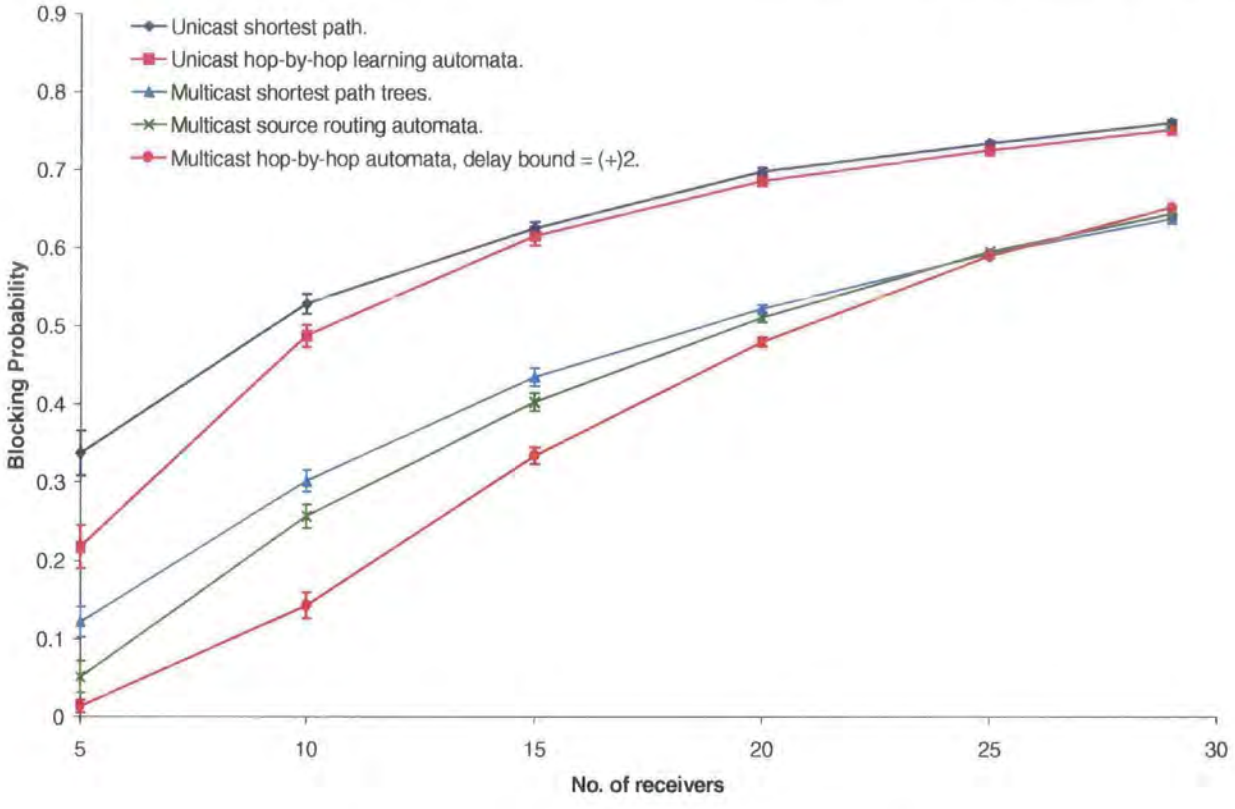


Figure 6.4 - Blocking probabilities, 20 groups.

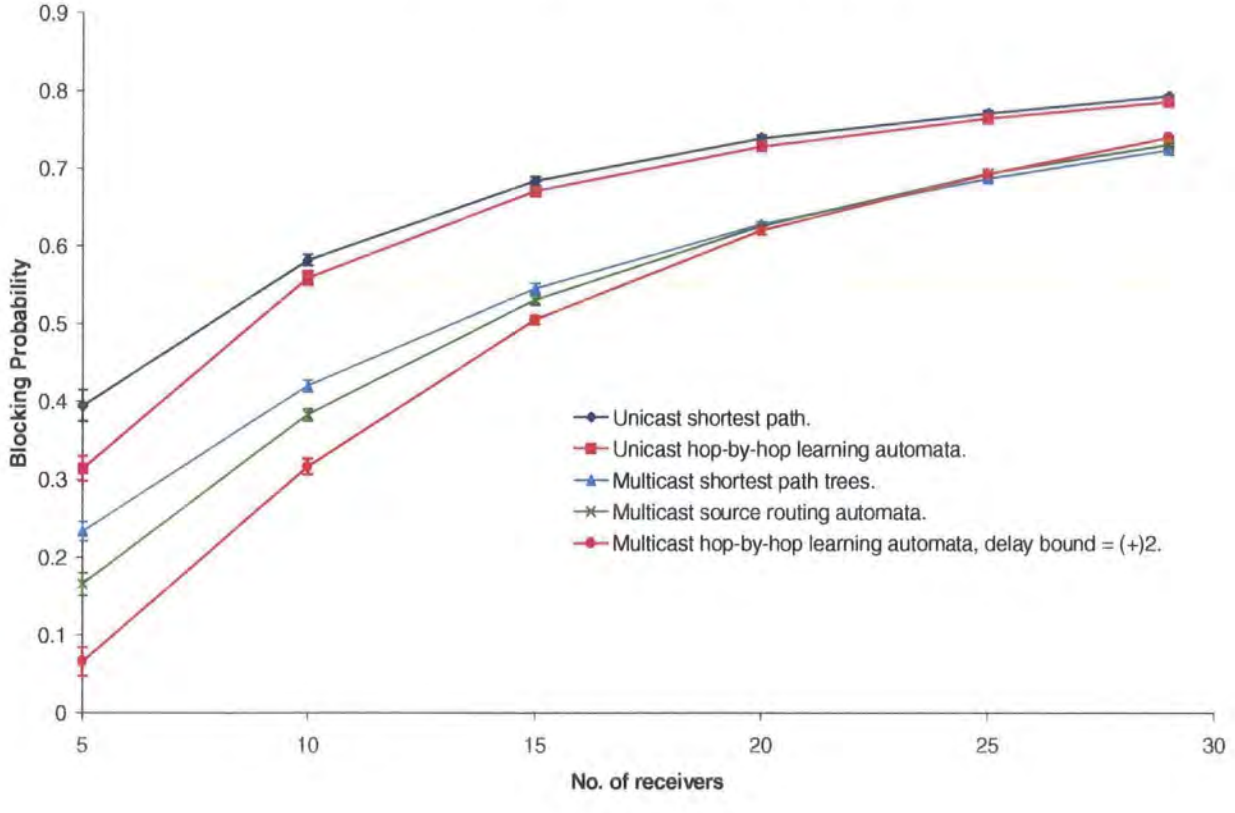


Figure 6.5 - Blocking probabilities, 25 groups.

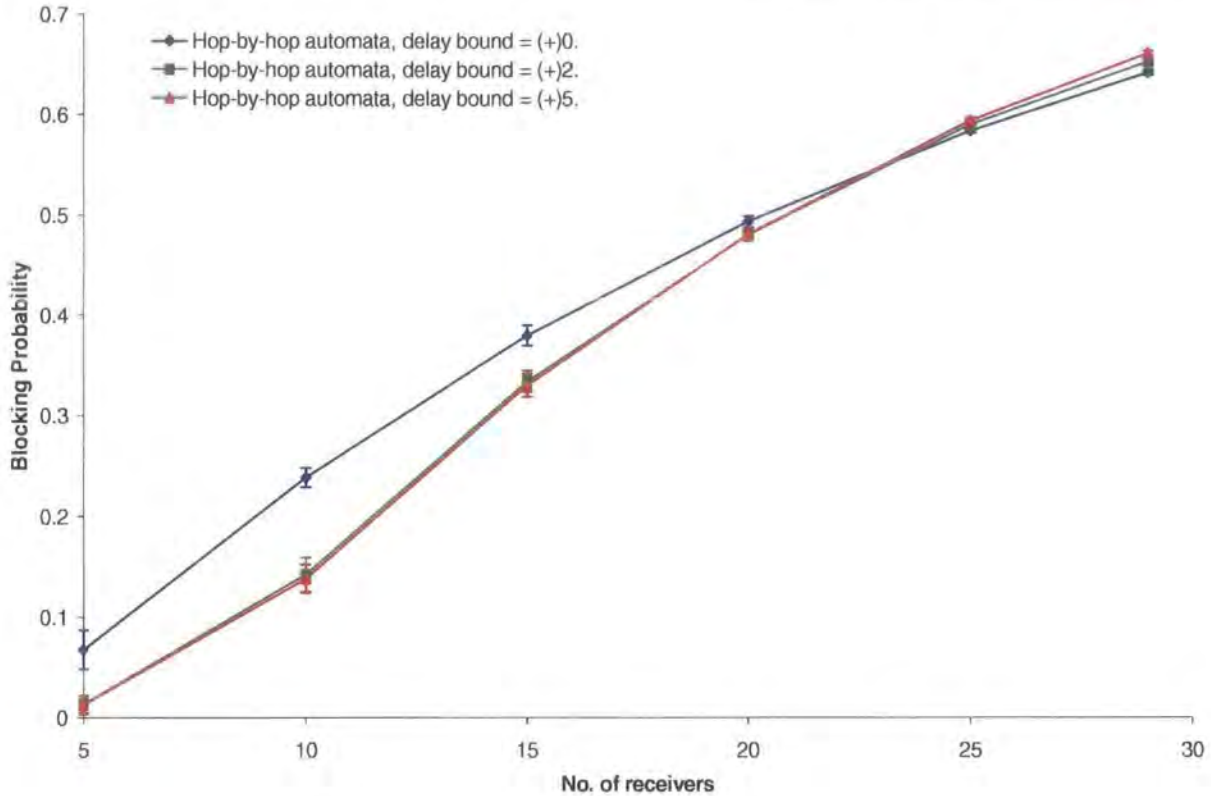


Figure 6.6 - Blocking probability, 20 groups, hop-by-hop automata.

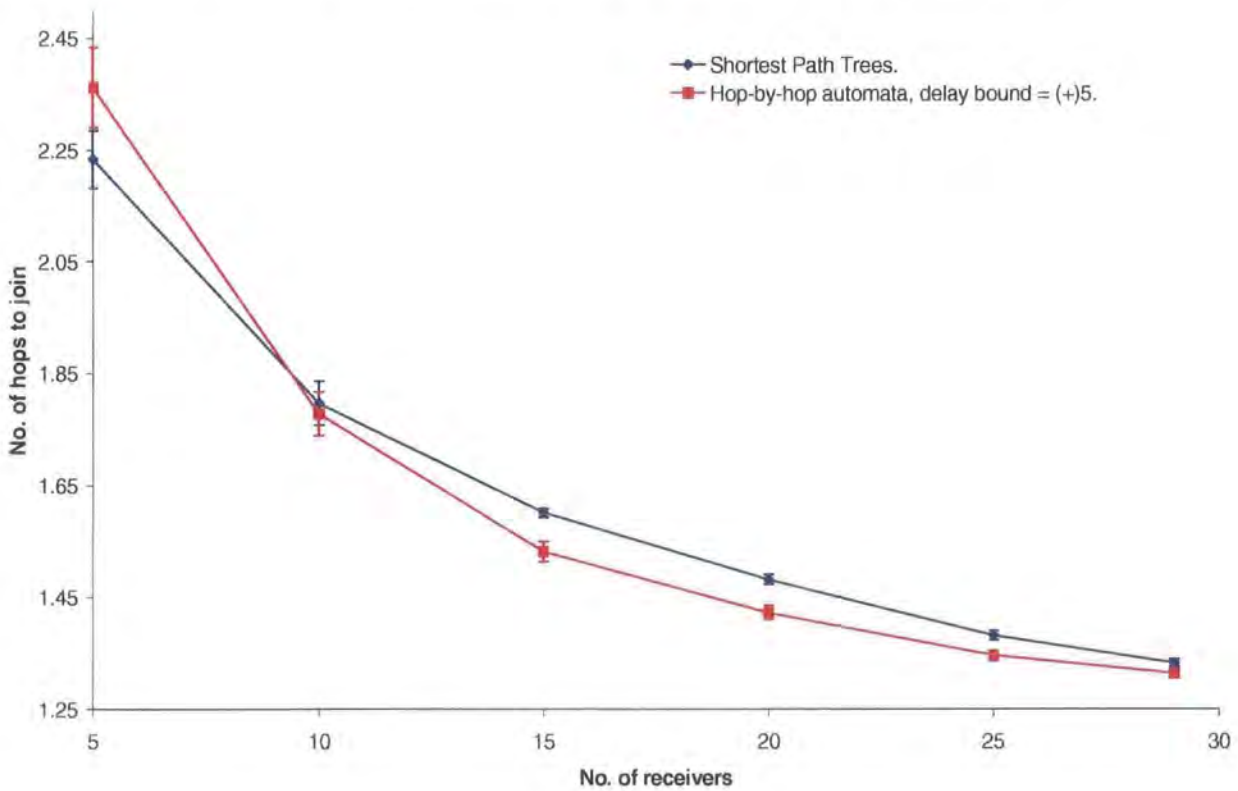


Figure 6.7 – Number of hops to join, 25 groups.

We see that for a low number of receivers per group, the potential for sharing is small and learning automata are taking longer routes to the source in order to load balance. As we increase the number of receivers per group, the potential for sharing increases and the automata join the groups in a lower average number of hops, therefore consuming a lower amount of resource on average in joining the groups. To show that automata are indeed learning to produce lower cost trees than shortest path routing, we let the automata converge, removed any contention (by increasing resources) and let all potential receivers join in a random order. We then measured the static cost of the resulting trees where cost is defined as the total number of nodes who are members across all groups. We repeated the same procedure for shortest path trees. We have also measured the static cost for shortest path unicast routing. Finally, we computed the optimal (unconstrained delay) Steiner tree cost using the package in [153]. In Figure 6.8, we show the static costs plotted against number of receivers per group for the 15 multicast group case, confidence intervals are removed for clarity. It can be seen that the learning automata for multicast routing are indeed producing lower (static) cost trees than shortest path routing, indicating that the automata have learnt which nodes are likely to be a member of a given group, therefore sending join requests on paths which are likely to meet existing members in the fewest number of hops. The automata with the tighter delay bound ((+)0 hops in this case) produce higher cost trees since they can only choose from all shortest paths to minimise tree cost, although they still reduce costs over a shortest path tree.

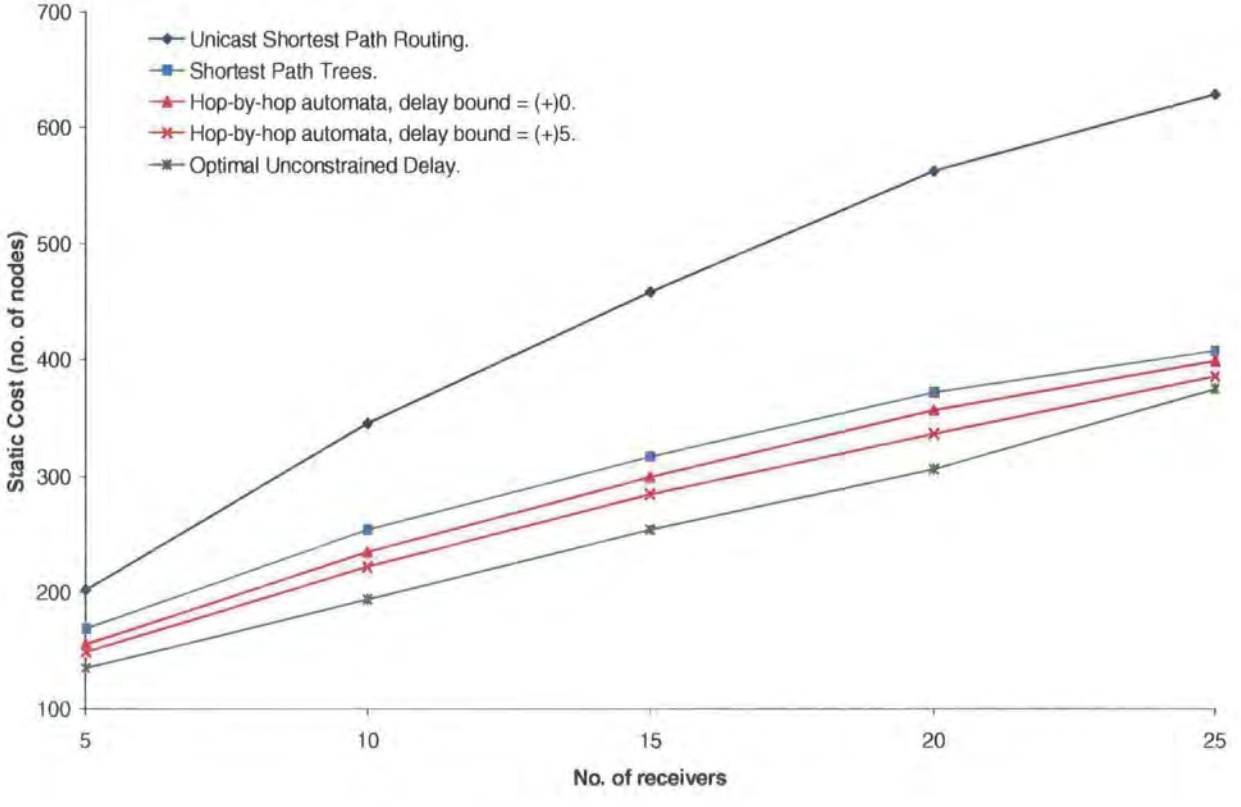


Figure 6.8 – Static Costs, 15 groups.

The cost of unicast routing diverges from multicast as group membership increases and the potential for sharing or merging of reservations grows.

We went on to measure the dynamic cost of the resulting multicast trees in the dynamic experiments, where we now define the cost of a multicast tree as the percentage of non-edge or non-receiver nodes as a fraction of the total number of member nodes across all groups. Since we are modelling a dynamic multicasting environment, the cost of the multicast trees in the network will change as the simulation progresses. We expect the average cost to reach a constant as the simulation reaches a steady state and the cost fluctuations become small. In Figure 6.9, we plot the steady state total cost against number of receivers per group for the 20 multicast group case, for shortest path, and hop-by-hop automata with delay bounds of 0 and 2. 90% confidence intervals are shown.

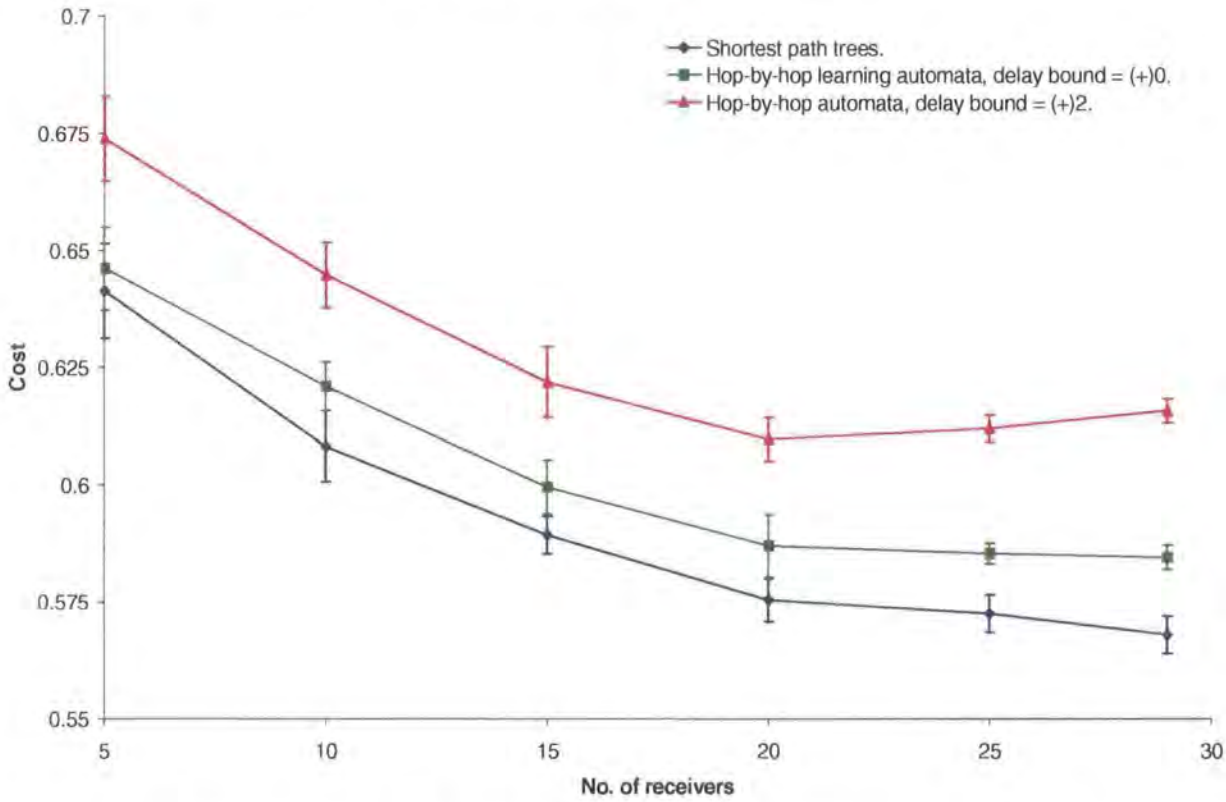


Figure 6.9 - Dynamic cost, shortest path and hop-by-hop automata, 20 groups.

It can be seen that the total cost of the hop-by-hop automata with bounds of 0 and 2 are slightly greater than that of the shortest path routing. This implies that the automata are constructing larger trees on average as they allow nodes further from the source to join through load balancing and sharing. The overall cost of the trees decreases with increasing number of receivers, since, as the blocking level increases, shorter low cost paths have a higher probability of acceptance than longer higher cost ones. This conclusion is confirmed in Figure 6.10, where we show sample paths of the average distance of nodes from the source receiving successful join acknowledgements for automata hop-by-hop routing with a delay bound of 2 and shortest path routing. For the automata, we also show the shortest path distance of

nodes from the source which represents the average radius of the multicast trees. For shortest path routing, the actual and shortest path distance from the source will be equivalent. The sample paths are taken from the 20 group, 10 receiver case. It can be seen here then that members are a greater average shortest distance from the source in the automata case showing that automata allow nodes which are further from the sources to join more frequently. The difference between the automata shortest (radius) and actual path lengths represent the degree to which automata are using longer alternate paths. We see according to the shape of the curve that automata converge reasonably well in around 200,000 join requests.

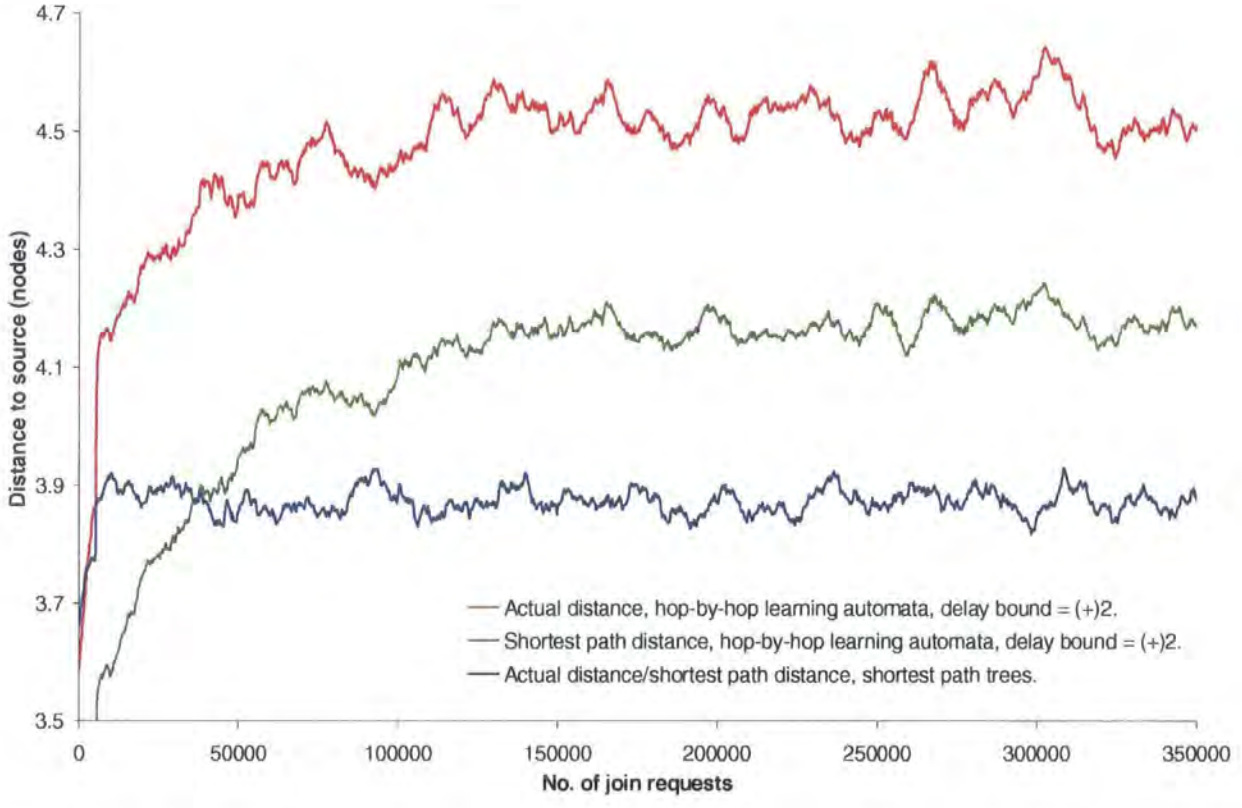


Figure 6.10 - Average shortest path distance to source, shortest path and hop-by-hop automata.

An examination of the total resources used in the network is shown in Figure 6.11 via three sample paths for the 20 group, 10 receiver case. We compare hop-by-hop automata with a delay bound of 2, source routing automata and shortest path trees. The graph shows that learning automata maximise the used capacity in the network through load balancing and sharing ability. This curve also suggests reasonable convergence in around 200,000 join requests, for the 20 group, 10 receiver case. Finally, we have examined the entropy of the automata probabilities for the two automata based routing schemes in Figure 6.12. We show sample paths of entropy for source routing and hop-by-hop automata for the 20 group, 10 receiver scenario, where entropy is plotted against number of join requests sent.

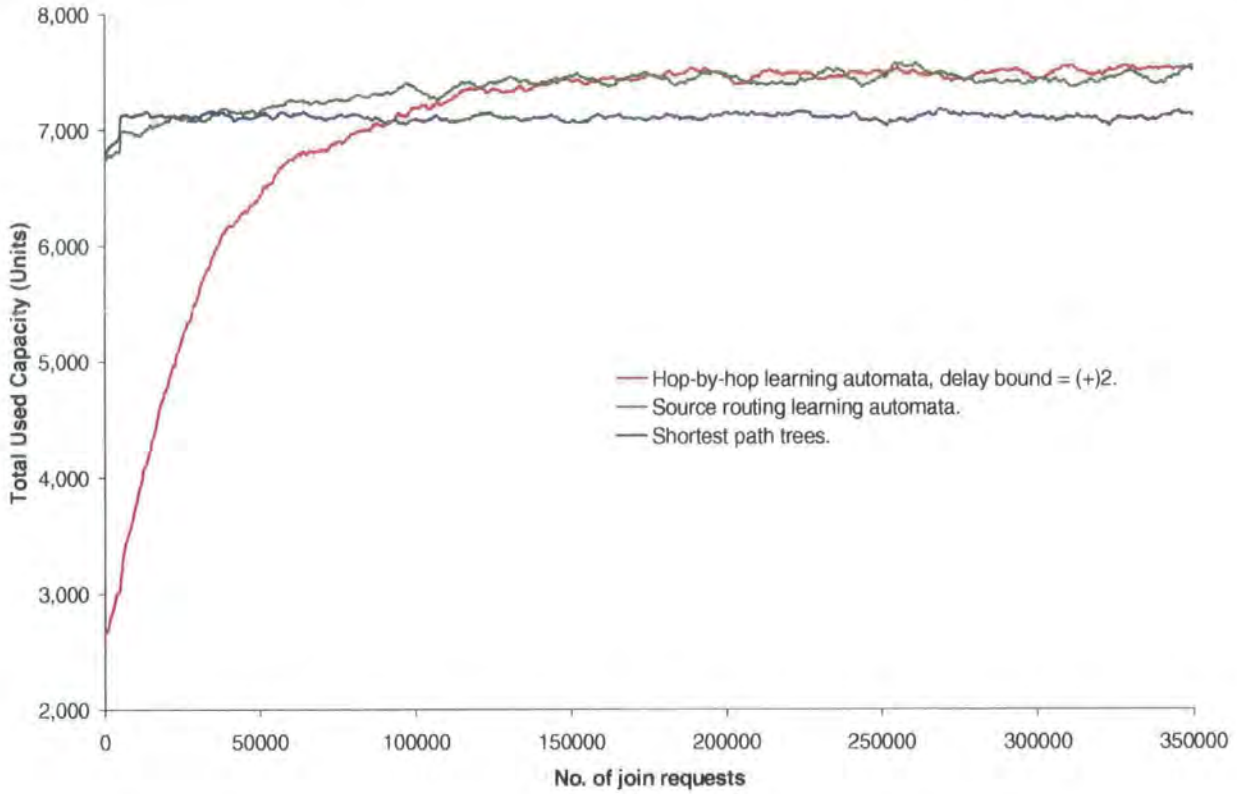


Figure 6.11 - Total Used capacity, shortest path and hop-by-hop automata.

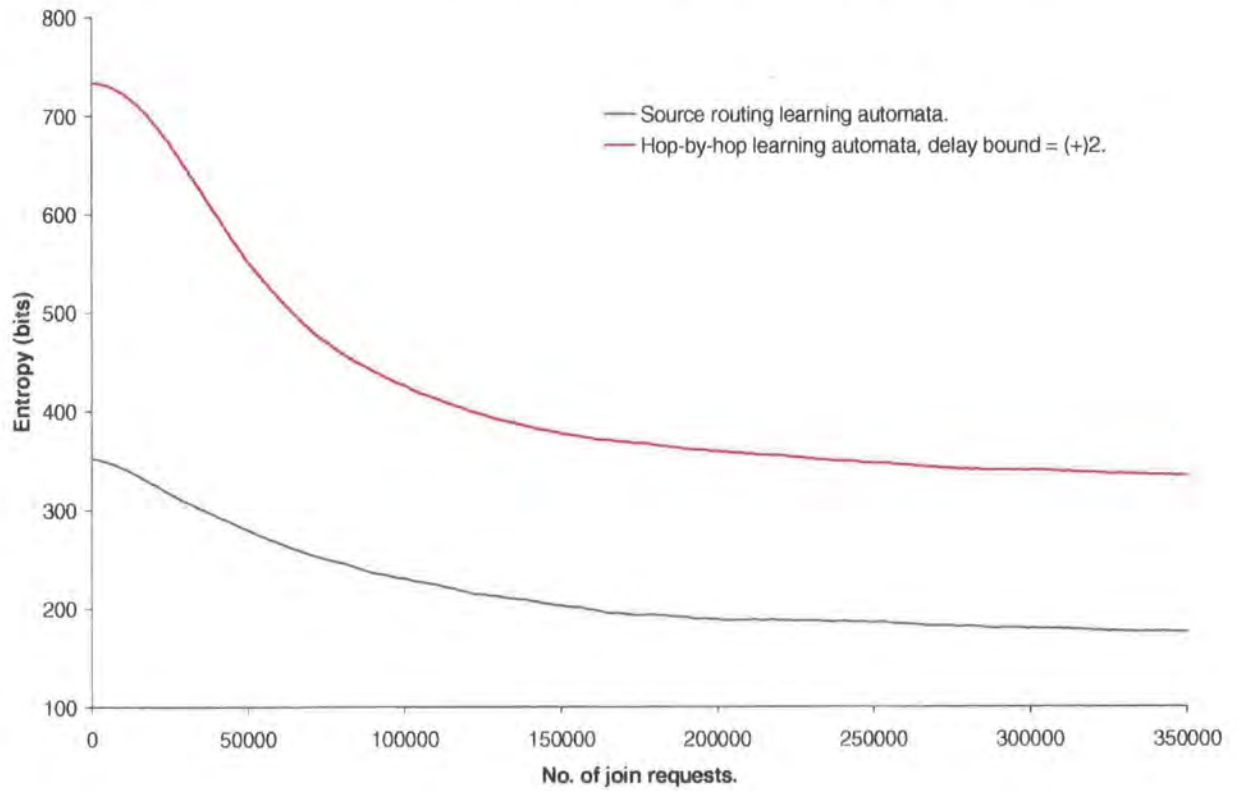


Figure 6.12 - Entropy sample paths, 20 groups 10 receivers.

The entropy plots of both automata schemes show that both automata schemes are converging reasonably well in around 200,000 join requests. The entropy of the hop-by-hop automata is higher than the source routing automata since the hop-by-hop automata have more actions on average, whereas the source routing automata all have two actions, representing two stored source routes.

Source routing has the advantage that we start the convergence process from already feasible routes but requires that we have global topological knowledge to calculate these routes. Hop-by-hop automata on the other hand start with no directly programmed routes but require only local connectivity information. In addition, the probabilities of the hop-by-hop automata could easily be programmed to point towards a feasible route set, reflecting any prior information known about the multicast environment.

6.7. QoS-bounded Shared Multicast Trees

The previous work has examined the application of learning algorithms to constructing per-source trees. Here, we examine how they may be used to create QoS-bounded shared trees. There has been very little previous work in this area, although [141] examines some possible centre selection algorithms for QoS-bounded shared trees. As we observed in the previous chapter, shared trees have considerable advantages over source routed trees. Specifically, it takes less overhead to construct and maintain one shared tree per multicast session than to construct a source-specific multicast tree for every source transmitting to that session [141]. When a receiver or source wish to join a shared tree, they simply forward their request on a path (as determined by routing) to the tree. They do not need explicit knowledge of the sources or receivers that already make up the tree. In general, a potential receiver or source that wishes to join the tree will only need to know the address of the core (centre) and the QoS that the shared tree may support. We know from Chapter 6 that the construction of centre based trees consists of two parts. Firstly, we must locate a core or topological centre and secondly, we must select the appropriate routes from nodes to that centre. Here, we are concerned with the second point. That is, given that we have a centre, how should we calculate paths from a source/receiver to the shared tree? In general, a shared tree does not have to have a centre but doing so eases many management functions. In particular, for the QoS-multicasting problem, the existence of a core or rendezvous point (RP) [138] enables us to bound the delay between any two nodes in the tree as follows. If any node in the tree is within $\Delta/2$ hops of the core (C), then any two nodes in the tree will be a maximum of Δ hops away from each other, as demonstrated in Figure 6.13.

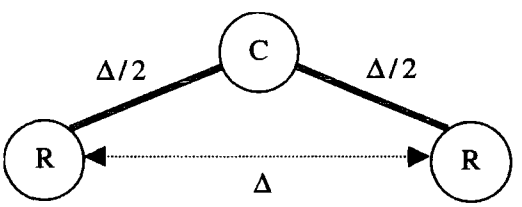


Figure 6.13 – Delay bound Δ between two receivers.

In the model that we consider, all nodes that join the tree are considered to be both sources and receivers, and all nodes make a single reservation to the tree. For example, assuming 20 nodes join the shared tree, and each node makes a reservation of 20Mbits/s, if all nodes send at the same time then they must share the reservation of 20Mbits/s. This is similar to the shared reservation style of RSVP [7]. In Figure 6.14, we depict a shared tree with a centre C, and 4 group members, D1 through D4. Nodes D1 and D3 are shown sending packets to the group. In this case $\Delta/2$ is 2 hops so that any two members on the tree are a maximum of 4 hops apart. We can see from Figure 6.14 that if $\Delta/2$ is equal to one hop, then only node D2 may join the multicast group. Thus, for extremely tight delay bounds, it may be impossible to construct feasible paths to meet the delay bound depending on the location of the core. To perform simulations for QoS-bounded multicast shared trees, we have used the simulation model created for source routed trees discussed previously. Now however, the source of the multicast group previously can be considered to be the centre of a shared tree. For example, for the 15 group, 15 receiver case, 15 different centres will be randomly located as will 15 receiver/sources for each shared tree using that centre. In addition, the delay bound $\Delta/2$, is specified as a maximum value rather than a certain number of hops greater than the shortest path length.

We have performed steady state simulations for the 15 group, 15 receiver/source case. In Figure 6.15, we plot blocking probability against $\Delta/2$ (hops) for shortest path routing and hop-by-hop automata. It is observed that for low hop-count bounds, both shortest path routing and automata based routing produce high levels of blocking since the majority of blocking events are due to delay bound violation rather than lack of spare capacity. As the hop-count bound is increased, the shortest path and automata blocking traces diverge as automata make better use of the available capacity in the network. The largest shortest path distance between any two nodes for the 30 node network studied is 7 hops, and we see that as the hop-count (delay) bound ($\Delta/2$) passes the 7 hop mark, the shortest path blocking remains constant since there are no failures at or beyond this point due to hop-count bound violation. The automata blocking trace continues to fall however as the automata make use of the longer alternate paths to maximise the amount of sharing and load balancing taking place. To support multiple levels of QoS, a shared tree could be constructed for each different QoS level, each tree associated with a different core or centre. Additionally, if tighter delay bounds are required between the sources/receivers, one large shared tree with lax bounds may be replaced with a number of smaller shared trees where the core/centre is in close proximity to the relevant groups of sources/receivers.

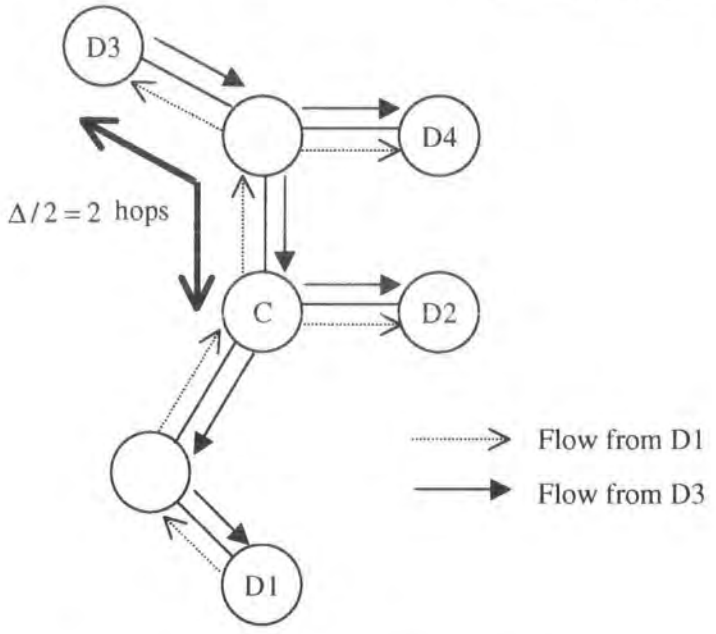


Figure 6.14 – Example of QoS-bounded shared tree.

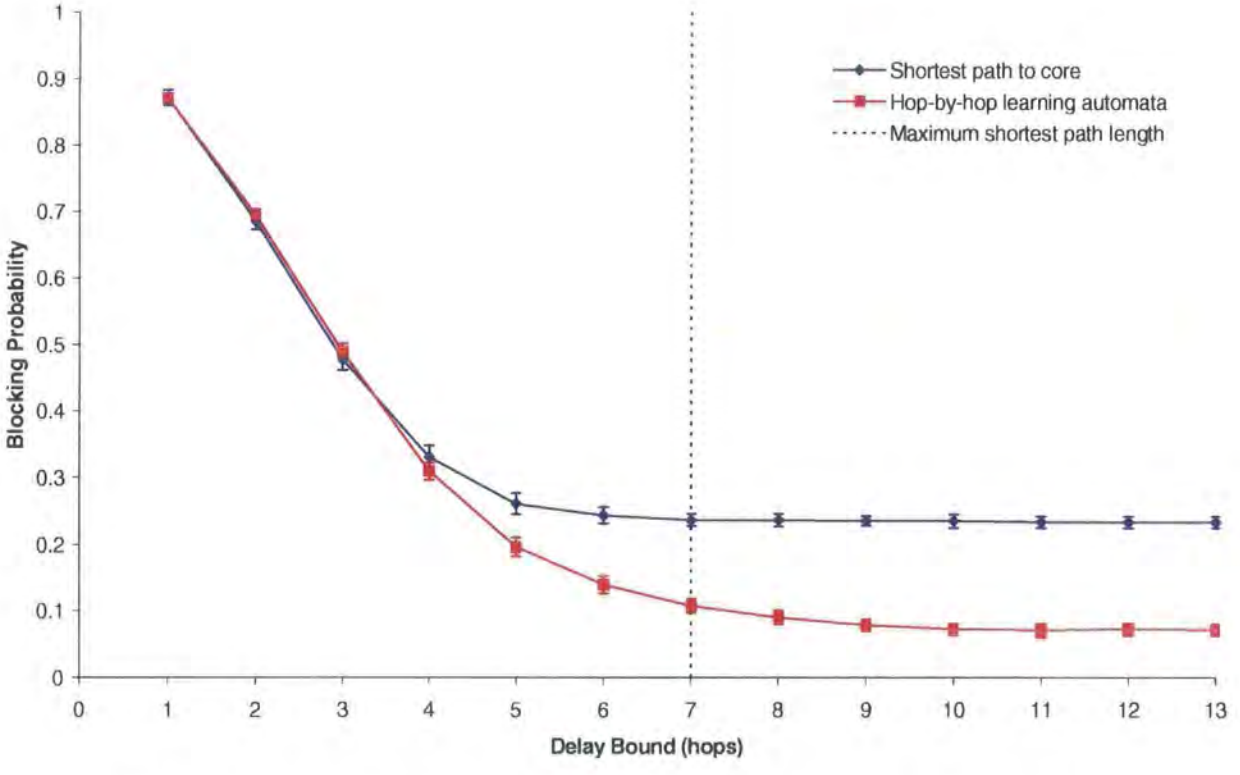


Figure 6.15 – Blocking Probability, QoS-shared trees, 15 groups, 15 receivers.

6.8. Combined Unicast and Multicast Routing.

Previously in this thesis, learning automata have been used to route both unicast and multicast reservation based traffics. Here, we examine the potential for one set of learning automata to route both unicast and multicast reservation requests on a single network. We compare this approach with maintaining discrete automata to route each traffic type. We also compare the approach with shortest path routing where reservation requests are simply routed on the shortest path to a destination regardless of the traffic type. Assuming the same topology and unicast traffic characteristics for the 30 node network as those examined in Chapter 3, and for even unicast traffic demands, we have plotted unicast and multicast blocking probability against unicast arrival rate for the case where there are 15 randomly located multicast groups each with 15 randomly located receivers. In Figure 6.16, we display plots for combined automata routing, discrete automata routing and shortest path routing. 90% confidence intervals are shown.

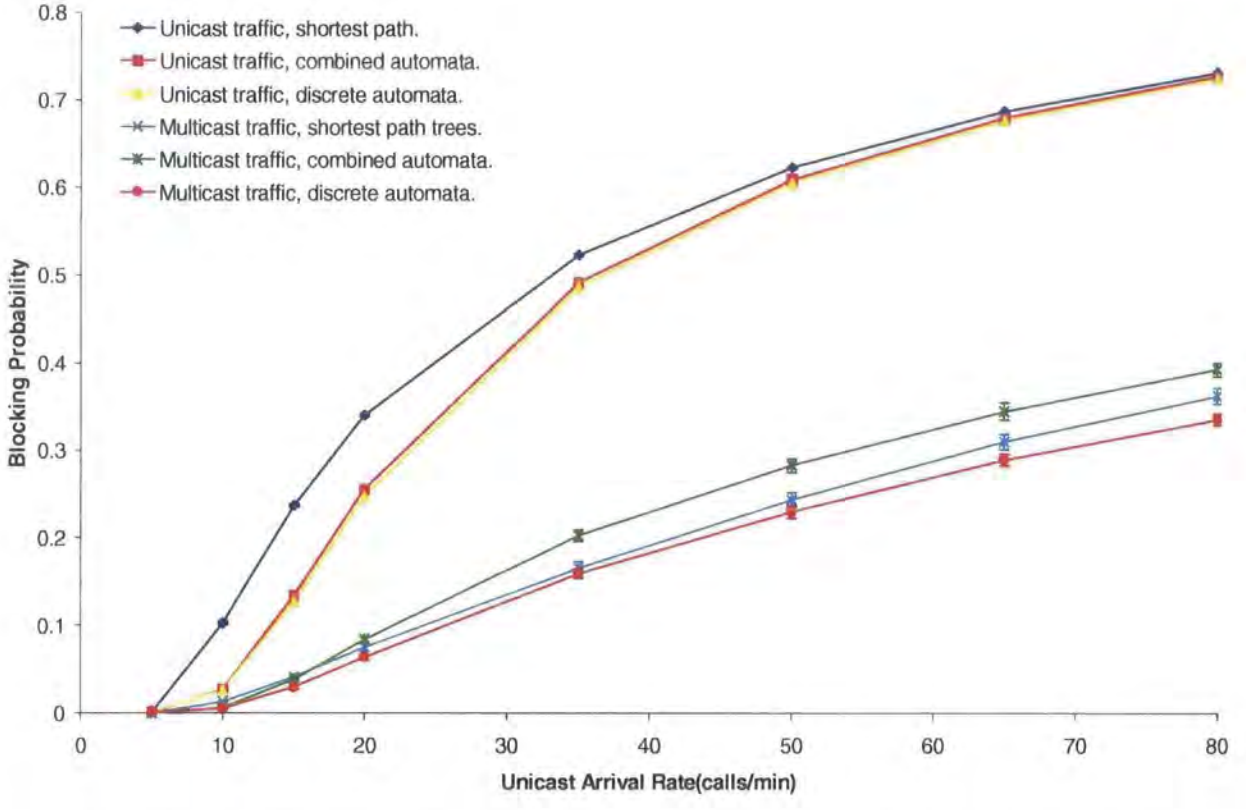


Figure 6.16 – Unicast and Multicast blocking probability for varying unicast arrival rate.

From a unicast perspective we see that using a single set of automata to route both traffics results in little degradation in blocking probabilities. However, for the multicast traffic, we see that the combined automata diverges from the discrete automata solution for high unicast arrival rates. This is the case since the combined automata are effectively converging to the unicast solution since the number of unicast events per unit time far outweighs the number of multicast events at high arrival rates. Although the unicast learning automata may load balance, the optimal multicast solution requires that we enable the

maximum amount of sharing to take place in addition to load balancing. This is confirmed in Figure 6.17, where we plot the number of hops travelled by join requests versus the unicast arrival rate for the three multicast routing mechanisms. 90% confidence intervals are shown.

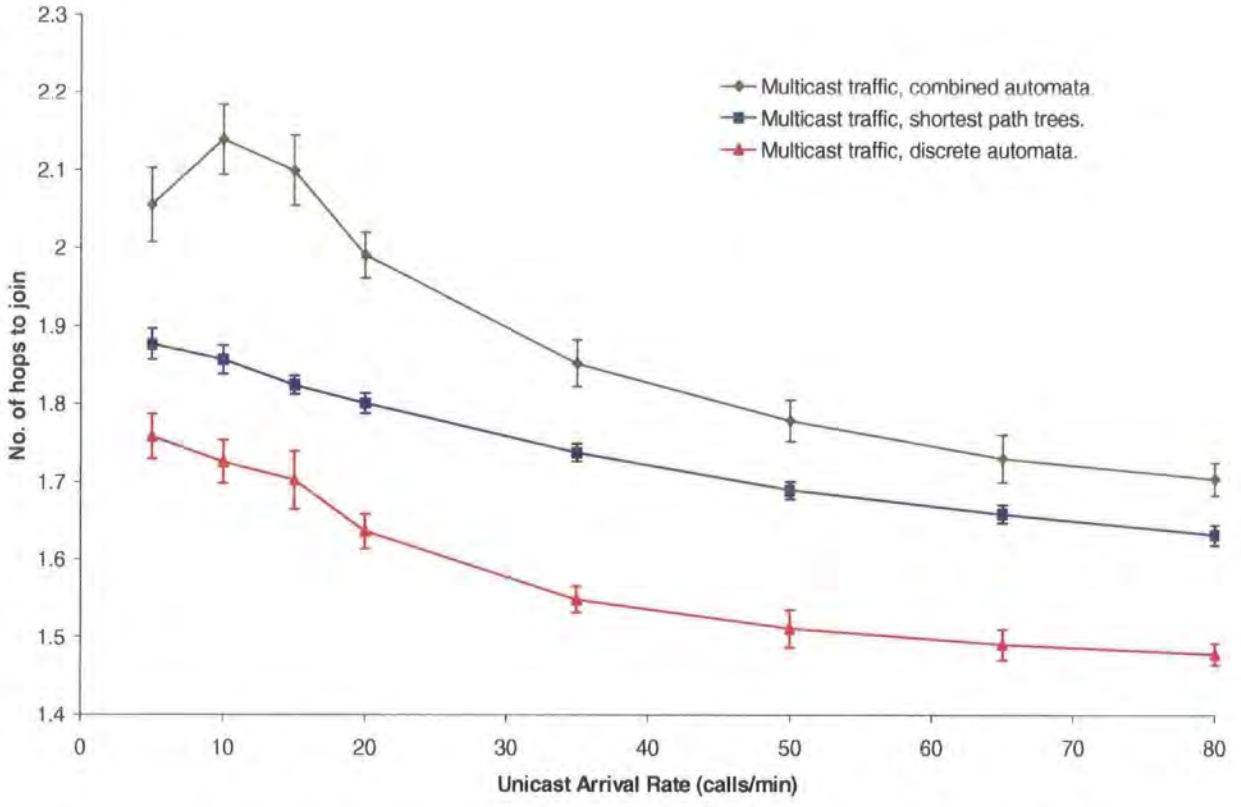


Figure 6.17 – Number of hops to join group(s), multicast routing algorithms, varying unicast arrival rate.

It can be seen that the discrete automata are learning to minimise resource consumption by selecting minimal hop paths to join the groups, whereas the combined automata scheme is selecting the longer alternate paths to meet the load balancing requirements of the unicast traffic causing the muticast traffic to take longer paths to join the groups. For the second experiment, we have kept the unicast traffic arrival rate held constant at 10 calls/min, whilst varying the number of randomly located receivers per group for the 29 multicast group case. In Figure 6.18, we plot unicast and multicast blocking probability against number of receivers per group for the three routing approaches shown previously. Now, the multicast traffic starts to dominate the traffic mix so that the combined automata will mainly converge to the multicast traffic requirements. Since the multicast automata will load balance as well as maximise sharing potential, the difference between the discrete and combined automata traces are minimal. For the shortest path routing, the blocking probability to the multicast traffic decreases as the number of receivers per group increases.

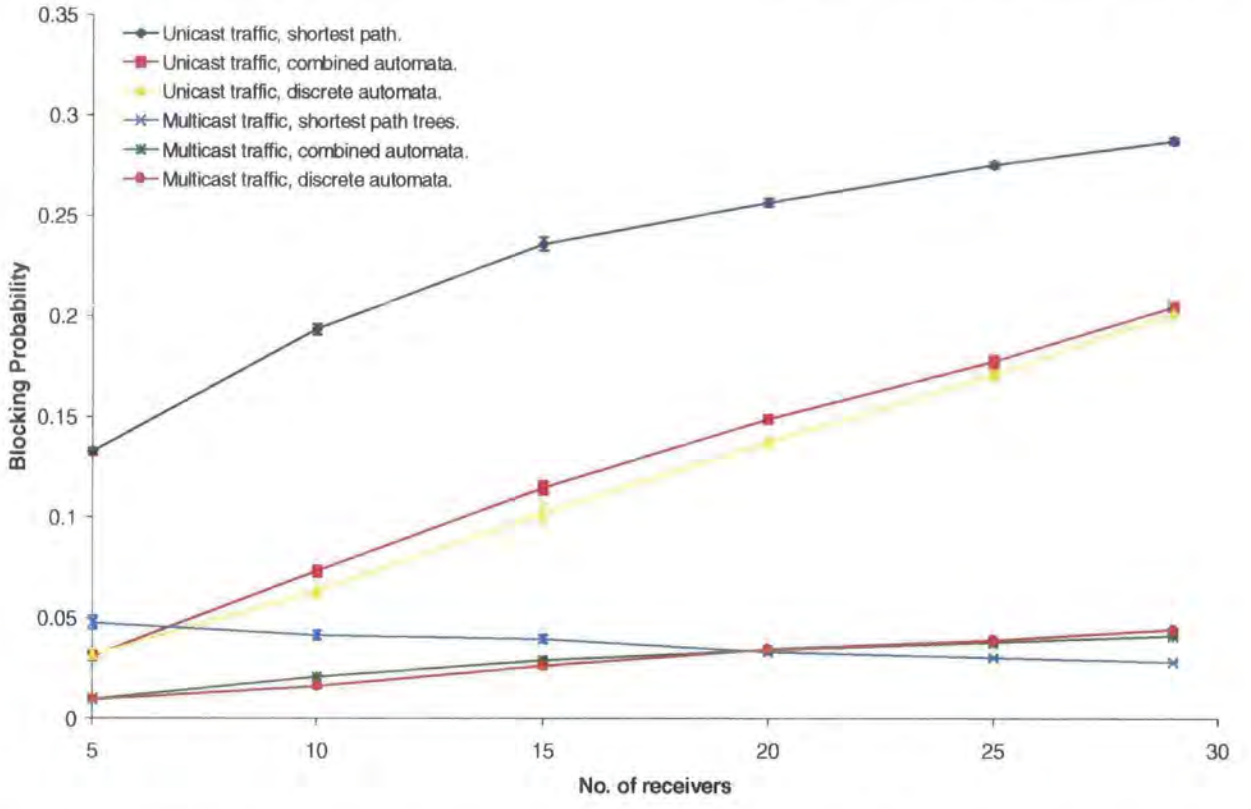


Figure 6.18 – Unicast and Multicast blocking probability for varying no. of receivers per group.

Since a multicast 'call' consumes less resource than a unicast 'call' on average, we can pack more multicast sessions on the shortest paths at the expense of the unicast traffic. Due to the statistical multiplexing effect, the blocking probability of the multicast traffic therefore drops at the expense of the unicast traffic. The effect is not prominent for the automata routing schemes since the automata spread the load across the network diminishing the effect, isolating the two traffic types to a greater extent.

6.9. Summary

In this chapter, we have studied novel algorithms for creating multicast trees capable of satisfying the Quality of Service (QoS) requirements of real-time applications for future high-speed integrated-service networks. Both per-source and shared trees have been examined. Simulations on a 30 node network showed that learning algorithms are capable of minimising blocking probabilities in a dynamic membership environment by learning to choose routes that maximise the use of available resources whilst meeting the required delay bounds of the trees, through sensible use of alternate paths. Learning is possible since there is a binary feedback response to a receiver requesting to join, telling it of the success or failure of its join request. We found that the learning algorithms created trees with significant improvements over traditional shortest path trees (SPT) which do not explicitly consider the sharing of resources within the network. Previous work on QoS bounded multicast trees has primarily focused on the construction of trees between pre-specified static members and so called constrained Steiner tree

(CST) heuristic algorithms have been proposed in this context. It is unlikely that these algorithms can be used in a practical dynamic membership environment such as that studied in this chapter however, due to their general requirement for global dynamic state information and their computational complexity. The learning algorithms suggested here work in a distributed fashion and require only limited topological information together with the binary feedback response specified above which is likely to exist in any future real-time multicasting protocol. Furthermore, the proposed hop-by-hop automata algorithms can operate just as well at the inter-domain level as the intra-domain level since we only require a node's local connectivity and knowledge of the multicast group address which we are trying to join.

The final section of this Chapter examined the routing of both unicast and multicast reservation based traffics on one network. We found that using one set of learning automata to route both types of traffic gives good overall blocking performance although can degrade performance to the minority traffic relative to using discrete learning automata for each traffic type. This effect is most pronounced for a unicast dominated traffic mix since the unicast learning automata learn to load balance but do not take account of the sharing requirements of the multicast traffic.

Chapter 7

Conclusions and Further Work

In this thesis, learning automata have been examined for adaptive routing in integrated service networks. Integrated service networks differ from circuit and packet switched networks in that they are expected to carry many different traffic types simultaneously, and we require adaptive control mechanisms that will scale to large networks with a potentially very large number of traffic flows. We have considered the use of learning automata for unicast and multicast routing of real-time and non-real-time traffics. The underlying assumptions have been that a dynamic state based routing strategy will not scale to large networks due to the need for a flooding process. In addition, state based routing strategies will not give significant advantage over quasi-static schemes when the state is changing in the network very rapidly.

In Chapter 1, the fundamental control mechanisms for supporting real-time traffic in integrated service networks were introduced. In addition, the general network control problem was formulated and it was concluded that adaptive control will play an important role in future networks due to the increasing uncertainty of the nature and volume of traffic on these multi-service networks. Furthermore, Artificial Intelligence (AI) techniques were proposed as potential adaptive control techniques since they can adaptively control systems without the need for a mathematical model of the system.

Chapter 2 carried out a broad review of the application of AI techniques to network control. For Fuzzy Logic and Artificial Neural Networks (ANNs) applied to network control problems, most of the techniques in the literature pose a slightly different problem to solve and there is a lack of comparison between the various models. Many of the papers do not make it clear what AI techniques can add to network control that traditional mechanisms do not. Much of the experimentation is performed with artificially generated traffic models (with high degrees of correlation) even though many real data traces are becoming available. There is a need then for the AI mechanisms proposed to be more rigorously compared with existing techniques, both theoretically and experimentally. Theoretically, the computation, communication and storage overheads should be derived. Experimentally, real data traces should perform the basis of comparison. Intelligent agents were identified as a promising area for future research, particularly reactive agents, which perform fast acting localised control using simple underlying rules. One interesting further work area would be to examine the potential of a subsumption (hierarchical) architecture for network control.

Learning automata were identified as an AI mechanism where the benefits of their application are clear. Automata provide a totally distributed (quasi) adaptive routing capability suited to systems where

we do not have knowledge of the incoming traffic demands and cannot afford to have access to dynamic state information due to overhead and/or policy restrictions. Although automata have been examined previously for routing in circuit and packet switched environments, the nature of integrated service networks warrants a re-examination of their potential application.

In chapter 3, learning automata were investigated for the routing of real-time flows, also known as the quality-of-service (QoS) routing problem. Here, the actions of the automata could correspond to a complete source route or simply the next link for a connection to be routed over. Automata represent an intuitively simple solution to QoS routing since if the chosen route meets the QoS requirements of the flow, the probability of selecting this route is increased whilst the probability may be decreased if the route does not meet these requirements. For the simulations in this thesis, the QoS requested by incoming flows consisted of a bandwidth and a path length bound parameter. This QoS set institutes a practical set from which, other QoS values of interest can be derived. The performance of learning automata routing was compared with shortest path routing in terms of blocking probabilities on two different network topologies. It was found that using one set of distributed learning automata gives considerable improvement in blocking performance, particularly in the mid-congestion region where the maximum benefit of load splitting is accrued. The number of paths used by the automata could be varied by altering the path length bound, tighter bounds improving blocking performance at high loads. When multi-rate traffic was presented to the networks, one set of learning automata were found to route the traffic as effectively as using automata per traffic class. Additionally, routing with automata based on source and destination address was found to give little benefit over a more traditional automata implementation with destination based automata alone. Automata were also applied to a simple hierarchical routing problem where the 30-node network was arbitrarily divided into 3 domains. Despite an information loss of approximately 2/3, the automata provided excellent blocking performance as compared to a shortest path routing approach with border nodes responsible for inter-domain routing. The advantage of load splitting was found to be diminished when the traffic demands consist of a small number of high bandwidth flows rather than many small bandwidth ones, due to a bandwidth fragmentation effect. Finally, automata were examined for a number of resource reservation models in high bandwidth-delay product environments, and it was found that the learning rates must be set carefully to avoid oscillations in the automata probabilities or the latching of the probabilities to 0 or 1.

In Chapter 4, automata were examined for the routing of NRT and mixed traffics. Datagram based automata were adopted using packet delay as feedback. For NRT routing, automata were found to give lower delays at high loads through load splitting, although gave slightly higher delays than shortest path routing at low loads, due to the finite size of the control packets which have been ignored in previous studies. A mixed traffic environment was also studied, where a resource reservation (real-time) based traffic had priority over the NRT traffic which had access to the remaining bandwidth. This type of model could be extremely important, since initially, it is likely that network providers will design the network to provide a specific QoS for the real-time traffic, and 'soak' up the remaining performance due

to statistical fluctuations in a NRT traffic element. Initially then, adaptive resource allocation mechanisms may be used primarily for NRT services. In the mixed traffic environment, it was found that using learning automata to route the RT traffic can improve upon shortest path routing in terms of blocking probabilities to the RT and average packet delay to the NRT. We investigated using the RT automata probabilities to route the NRT traffic and found that the average delay to the NRT depends on the relative RT and NRT traffic distributions and the level of load splitting being carried out by the RT traffic.

In Chapter 5, it was demonstrated that learning automata could be applied to the multicast routing problem. In a receiver-oriented dynamic multicast environment where receivers are continually joining and leaving the multicast group, the automata were shown to minimise either the average received packet delay or the total tree cost, where the cost was defined as the total number of multicast group members. To enable the minimisation of delay, the feedback to the automata was the average packet delay over a connection's duration. To minimise total tree cost, the feedback to the automata was the number of hops travelled to join the multicast tree. Thus, the automata converged to minimise the number of hops taken to join the tree and therefore the resources consumed by the tree. Automata applied in this way were effectively shown to behave as a minimum Steiner tree heuristic suitable for dynamic environments. Existing heuristics typically require global knowledge of the link costs/delays and perform a centralised computation. Automata on the other hand require only local connectivity knowledge together with a simple updating strategy, enabling a totally distributed computation.

In Chapter 6, the use of automata for multicasting was extended to dynamic QoS bounded environments, where the QoS required by receivers was the bandwidth required together with a path length constraint to the source of the group (as for Chapter 3). Automata used a simple binary feedback giving notification of the success or failure of a join to a multicast group. It was shown that automata minimise blocking probabilities over a shortest path tree approach through a combination of load balancing and creating low cost multicast trees. The approach was also shown to be viable for QoS shared multicast trees, where sources and receivers now have a path length bound to the centre of the shared tree rather than to the source of the group as for per-source multicast trees. Finally, we investigated the case where unicast and multicast reservation based traffics are contained on one network. It was found that one set of learning automata can route both reasonably effectively, although the blocking performance to the minority traffic is degraded relative to using separate learning automata for each traffic type.

7.1. Further Work

There are a number of potential areas for further work. In terms of the application of distributed learning automata, the question of convergence still remains an outstanding issue. In particular, the rate of convergence for automata in routing studies should be derived as a function of the size of the network (N). In practice however, it is extremely difficult to obtain real insight into the rate of convergence for

even trivial example networks. One step forward would be to carry out an empirical study, where a large number of networks of varying size and connectivity are simulated with basic traffic demands. Whilst the rate of convergence of automata in stationary environments is well understood, there is little theory regarding their rate of convergence in non-autonomous, non-stationary environments. Ongoing work at Durham [160] is attempting to compare the rate of convergence of different reinforcement learning algorithms in a simple 2-path routing problem similar to that demonstrated in Chapter 2. Once we obtain additional knowledge on the rate of convergence for the set of distributed automata, it should be possible to derive the computational complexity of an automata implementation.

The learning automata examined in this thesis have operated totally independently of one another. There is possibly some scope for the co-operation of automata, and to examine the effect of this inter-automaton communication in terms of transient and steady state behaviour. One of the attractive points of automata from a routing perspective however, is that automata do not require a communication process between the nodes (i.e. they have no communication overhead). If we introduce a communication process between the nodes, we should compare automata routing with other routing schemes that require message passing in the network.

There is a need for automata based routing to be studied on a real network with real traffic demands. In this way, it should be possible to compare the overheads of learning automata with a practical shortest path routing algorithm. In terms of practical experiments in this area, 'free' versions of UNIX exist which are well supported (e.g. FreeBSD, Linux). Unlike many commercial operating systems, it is possible to get at the code in these free versions of UNIX. Furthermore, FreeBSD is the preferred experimental platform for the IETF so that many of the initial versions of proposed protocols are freely available for the BSD platform (e.g. RSVP, CBT etc..). The first step to enabling a test-bed network suitable for routing experiments is to get one single router operational. This would involve changing the underlying FIFO queuing strategy to support the classification and placing of packets in different priority queues. Software to do this is also available (CBQ - Class Based Queuing). Once one router is operating satisfactorily, a number of routers could then be connected in a simple topology. A typical PC has support for about 4 Ethernet cards, so that reasonably well connected topologies could be constructed. To enable experimentation with routing of resource reservation based traffic, a resource reservation protocol is required. RSVP (Resource Reservation Protocol) is freely available for download. The current version of RSVP basically assumes an underlying shortest path routing protocol. To experiment with routing algorithms that route traffic on multiple paths, the code of RSVP may also need changing. A large proportion of the work involved in this project may involve creating the processes that gather relevant statistics from the test-bed and create traffic inputs to the network.

In Chapter 4, learning automata were considered for the routing of NRT traffic. An 'open-loop' traffic model was assumed where there was no flow/congestion control. The next step for these studies is to consider the interaction of the adaptive routing with the flow control. Simple models of TCP are readily available and could be incorporated into the simulations.

In Chapter 5, automata were considered for the minimisation of average received packet delay for a single multicast group with one and multiple sources. Previous work on shared trees for the multiple group case has shown that CBT (core-based-trees) can lead to traffic concentration close to the core as many different traffic flows traverse the same links [145] (see Figure 5.4, Chapter 5). An interesting line of experimentation would be to examine the behaviour of a delay based automata implementation with multiple multicast groups, since the automata should learn to avoid traffic 'hot spots' that lead to extra delays. The minimisation of cost was also considered and it was shown that automata effectively acted as a distributed Minimum Steiner Tree (MST) heuristic suitable for dynamic environments. For multicast trees, there is a trade-off between the cost and average (or maximum) end-to-end delay of the tree. To produce a tree with the relevant trade-off between cost and delay, a combined delay/cost metric could be investigated for the learning automata approach.

In Chapter 6, automata were considered for the QoS multicast routing problem, where all receivers were assumed to have homogenous (bandwidth and delay) requirements. A natural extension to this work is to consider receivers with heterogeneous requirements. For the heterogeneous receiver case, when resource reservations are merged at junction points, each junction will reserve adequate resources for the most demanding receivers (if they pass CAC) and reuse them to support the less demanding ones [139]. Some receivers will be able to support higher QoS than others, so there could be a high degree of blocking due to the asymmetry in receiver QoS requirements. It may be better to have a separate multicast group (or tree) for each QoS level.

With adaptive routing, the problem is essentially one of directing user traffic on the appropriate paths to a specific resource to enable load balancing. For example, given a video server in the network, we can route traffic to this server on the appropriate paths to perform load balancing. Alternatively, we could move or replicate the video server to achieve the same effect. One relevant application of learning algorithms is to examine their potential for location of resources in the appropriate place in the network. This spans issues such as the location of repositories, mirrors and the core selection process for shared multicast trees. There could be interaction between the operation of an adaptive routing protocol and adaptive resource location, so that their respective timescales of operation should be determined carefully.

In Chapter 2, an agent structure was proposed (see Figure 2.5, Chapter 2) to tackle the network control problem at multiple spatio-temporal levels. Here, Learning Automata formed one of the layers in a multi-layer control structure where all layers are concurrently operating and mediated by a set of control rules. Similarly, learning automata were operated as a background learning process in Chapter 3. The performance of the overall control structure will depend on the interaction of these layers (e.g. resilience and adaptive routing). Questions here are, 'what information should be passed between the layers, and how should the control rules be derived?' Specifically, should the control rules make decisions based on local or global information, and how often should this information be gathered? Further work is required on hierarchical control models such as this where there is interaction between the different control layers.

Appendix A

Learning Automata - A brief overview

A.1. Introduction

A learning automaton can be defined simply as a device whereby, 'A finite number of actions can be performed in a random environment' [72]. With stochastic learning automata (SLAs), each action is selected with some probability, and based on the random response from the environment, the action probabilities are updated using a particular rule. Typically, an automaton interacts with the environment as shown in Figure A.1 below.

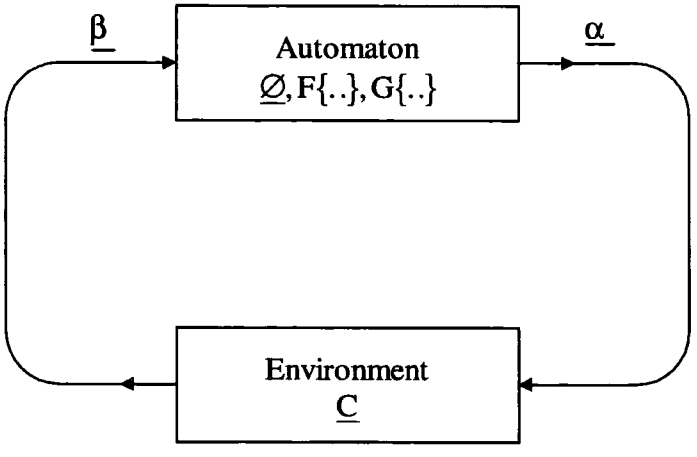


Figure A.1 - Automaton/Environment configuration

An automaton is defined by its state set $\underline{\varnothing}$, an output or action set $\underline{\alpha}$, an input set $\underline{\beta}$, a transition function $F\{..\}$ which determines the state at instant (n+1) in terms of the state and input at instant n, and an output function $G\{..\}$ which determines the output of the automaton at any instant n in terms of the state at that instant. Similarly, the environment can be defined by the triple $\{\underline{\alpha}, \underline{\beta}, \underline{c}\}$, where $\underline{\alpha}$ is the input set (output set of the automaton), $\underline{\beta}$ is the output set (input set of the automaton) and $\underline{c}$ is a set of penalty probabilities. The particular type of automata of interest for this work are 'stochastic variable structure automata' where the mappings F and G may be stochastic and the probabilities of actions can change at each iteration and are defined as follows for a r-action automaton. For a automaton A at instant n :

$$A(n) = \{\underline{\alpha}, \underline{\beta}, \underline{p}, T(\alpha, \beta, \underline{p})\}. \tag{A.1}$$

Here, we have an action set $\underline{\alpha}$ with r actions given by,

$$\alpha(n) \in \{\alpha_1, \ldots \alpha_r\}, \tag{A.2}$$

a response set $\underline{\beta}$ and a probability set $\underline{p}$ with r probabilities giving the probability of each action representing the internal state ($\underline{\varnothing}$) of the automaton so that,

$$\underline{p}(n) = \{p_1 \ldots p_r\}, \tag{A.3}$$

where $p_i = \text{prob}[\alpha(n) = \alpha_i]$, and the probabilities of the automaton sum to 1. The probabilities are usually initialised to be equal to one another (i.e. $p_i = 1/r, \forall i$). Prior knowledge can be contained within the probability set $\underline{p}$. For example, for the routing problem, if we have knowledge of the shortest path routes, the probability set $\underline{p}$ can be initialised to point towards the shortest path routes. A reinforcement algorithm T, provides the necessary means to modify the action probability vector with respect to the performed action and the received response. So that at instant (n+1), the probability vector can be written as :

$$\underline{p}(n+1) = T\{\alpha, \beta, \underline{p}(n)\}. \tag{A.4}$$

Finally, the environment is described by the triple E (n) as before,

$$E(n) = \{\underline{\alpha}, \underline{\beta}, \underline{c}\}, \tag{A.5}$$

where $\underline{\alpha}$ represents the input set of the environment, $\underline{\beta}$, the response set and $\underline{c}$ the penalty set. Further classification of the environment leads to a range of response models. A commonly used model is the P-model where the response to an action is binary so that $\beta(n) = 0$ is a reward and $\beta(n) = 1$ is a penalty. If the response is continuous in the region (0,1), then the model is called an S-model (i.e. $\beta(n) \in \{0,1\}$). The intermediate case is the Q-model where the response can take a finite set of discrete values in the region (0,1). Both P-type and S-type models are used in this thesis for the routing of connections where the chosen route is either a success or failure, and the routing of packets where the response in a continuous delay value and the delay is transformed into the region (0,1). The penalty set $\underline{c}$ (for the P-type model), dictates the probability that a given action will receive a penalty response $\beta(n) = 1$ and consists of r probabilities,

$$\underline{c} = \{c_1, \ldots c_r\}, \tag{A.6}$$

where $c_i = \text{prob}[\beta(n) = 1 | \alpha(n) = \alpha_i]$.

A.2. Performance Measures

A stationary environment can be defined as one where the set of penalty probabilities, $\underline{c}$, remain constant for all time. Non-stationary environments are conversely characterised by a penalty set $\underline{c}$ that varies with time. A special case of a non-stationary environment in the 'non-autonomous' environment where the

actions chosen by the automata, $\underline{\alpha}$, influence the values of the penalty set $\underline{c}$. This is the case for routing problems where routing connections or packets along some route decrease the attractiveness of selecting that route in the future. The following performance measures can be defined for the behaviour of a learning automaton. At stage n, if the action α_i is selected with probability p_i, the expected penalty is:

$$M(n) = E\{\beta(n) \mid p(n)\} = \sum_{i=1}^{r} p_i(n) c_i . \tag{A.7}$$

If the actions of the automaton are initially selected with equal probability, the value of the average penalty M_0 is given by:

$$M_0 = \frac{1}{r} \sum_{i=1}^{r} c_i . \tag{A.8}$$

In order to do better than a 'pure chance' automaton, an automaton must reduce its expected penalty, M (n), below M_0. A learning automaton is said to be 'expedient' if:

$$\lim_{n \to \infty} E[M(n)] < M_0 . \tag{A.9}$$

A learning automaton is therefore expedient if it does better than a scheme which chooses actions in a purely random fashion. A learning automaton is said to be 'optimal' if:

$$\lim_{n \to \infty} E[M(n)] = c_l , \tag{A.10}$$

where $c_l = \min_i \{c_i\}$. This condition means that the probability of choosing the action corresponding to the minimum penalty probability converges to 1. Since some forms of automata have 'absorbing states' such that we become locked with probability 1 into the optimal action, a learning automaton is said to be 'ε -optimal' if:

$$\lim_{n \to \infty} E[M(n)] < c_l + \varepsilon . \tag{A.11}$$

This means that the automaton can learn to choose the optimal action with a probability arbitrarily close to 1 with sufficient choice of ε.

A.3. Reinforcement Algorithms

The reinforcement algorithm T, modifies the action probability vector $\underline{p}$ (n) with respect to the performed action α (n), and the received response β (n) to give $\underline{p}$ (n+1). If $\underline{p}$ (n+1) is a linear function of $\underline{p}$ (n), the updating scheme is termed linear, otherwise it is non-linear. The state space of $\underline{p}$ (n) may be partitioned such that $\underline{p}$ (n) is updated using different schemes depending where the value of $\underline{p}$ (n) lies. Such a scheme is known as a 'hybrid' updating scheme. Non-linear reinforcement schemes have been investigated (see [72]) but gave no appreciable improvement over the linear updating schemes.

A general linear algorithm can be defined as follows for P-type environments. If $\alpha(n) = \alpha_i$, then :

If $\beta(n) = 0$ (favourable response) :

$$p_i(n+1) = p_i(n) + a[1 - p_i(n)]$$

$$p_j(n+1) = (1-a)p_j(n) \ ; \ \forall j \ ; \ j \neq i \, .$$

If $\beta(n) = 1$ (unfavourable response):

$$p_i(n+1) = (1-b)p_i(n)$$

$$p_j(n+1) = \frac{b}{r-1} + (1-b)p_j(n) \ ; \ \forall j \ ; \ j \neq i \qquad \text{(A.12)}$$

where **a** and **b** are the reward and penalty parameters respectively. If **a** = **b** in (A.12), the updating scheme is known as the 'Linear Reward Penalty (**LRP**) Scheme'. If **b** = 0, the 'Linear Reward Inaction (**LRI**) Scheme' is obtained. If **b** << **a**, (typically ten times less) the scheme is known as the 'Linear Reward epsilon Penalty (**LReP**) scheme'. In a similar manner, a general linear reinforcement scheme can be defined for S model environments. If $\alpha(n) = \alpha_i$:

$$p_i(n+1) = p_i(n) + a(1-\beta(n))(1-p_i(n)) - b\beta(n)p_i(n)$$

$$p_j(n+1) = p_j(n) - a(1-\beta(n))p_j(n) + b\beta(n)\left[\frac{1}{r-1} - p_j(n)\right] \ ; \ \forall j \ ; \ j \neq i \qquad \text{(A.13)}$$

where $\beta(n)$ is the normalised failure response of the environment at stage n. As for the P-model environment, we can define three linear reinforcement schemes. These are the **SLRI**, **SLRP** and **SLReP** for **b** = 0, **a** = **b** and **b** << **a** respectively.

A.4. Behaviour of Reinforcement Algorithms in Stationary Environments

For the above reinforcement algorithms operating in stationary environments (penalty set, $\underline{c}$, is fixed), the LRP scheme leads to expedient behaviour of the automaton, while both LRI and LReP schemes result in ε optimal (and expedient) behaviour. Additionally, the LRP and LReP schemes are 'ergodic' in that they converge in distribution to the optimal action probability vector independent of the initial action probability distribution. For non-ergodic schemes such as the LRI algorithm, the automaton has 'absorbing states' where the probability vector can become locked into a particular value for all time. Also, the state to which the probability vector converges is dependent on its initial value. The probability of converging to the optimal action can be made arbitrarily close to 1 by selecting smaller and smaller values of the reward parameter, **a**. Through the correct choice of reward and penalty parameters, the LReP scheme is both ε optimal and ergodic such that we will be able to adapt to changing penalty sets ($\underline{c}$), and we will be guaranteed to converge within a small tolerance to the optimal action. A scheme such as the LReP algorithm is therefore most suited to practical implementation in a non-stationary environment such as routing in a communications network.

A.5. Behaviour of Reinforcement Algorithms in Non-stationary Environments

In [79], Narendra *et al* have used non-stationary models of environments to attempt to model the steady state behaviour of learning automata routing schemes in communication networks. For the first model proposed, when an action α_i is performed at stage n, the corresponding penalty probability c_i of the environment increases while c_j $(i \neq j)$ decreases. For the second model, the penalty probabilities c_i are assumed to be monotonically increasing functions of the probabilities p_i with which the actions are chosen [79] (i.e. $c_i = f_i(p_i)$). Under certain conditions (see [79]), it can be shown for an LReP scheme that the probabilities assume values close to a unique equilibrium point p_i^* such that

$$f_1(p_1^*) = f_2(p_2^*) = \ldots = f_r(p_r^*) \tag{A.14}$$

and for an LRP scheme such that

$$p_1^* f_1(p_1^*) = p_2^* f_2(p_2^*) = \ldots = p_r^* f_r(p_r^*). \tag{A.15}$$

Thus, the LReP (and LRI) scheme tends to equalise the penalty probabilities whilst the LRP scheme tends to equalise the penalty rates. When used for circuit switched routing where blocking of a connection is the feedback response, this implies that the LReP scheme tends to equalise the blocking probabilities along the chosen paths whilst the LRP scheme will equalise the blocking rates. In packet switched networks where delay is fed back from the environment, we expect the LReP scheme to equalise delays along the various paths whilst the LRP scheme will equalise the delay rates. These theoretical results have been validated by simulation on simple networks in [79], [81], [85].

A.6. Other Reinforcement Algorithms

The reinforcement schemes described above are the most popular algorithms for the majority of applications where results are tractable for stationary environments. Additional algorithms include discretised algorithms where the values of the action probabilities are limited to discrete values in the interval (0,1). The idea is to approach an optimum directly rather than asymptotically as with continuous algorithms, and the trade off between rate of convergence and steady state accuracy can be controlled by varying the degree of quantisation. Also, discretisation enables the probabilities to be stored as integer values which could prove useful for practical applications where probabilities would otherwise be stored as floating point values, thus introducing truncation errors. Theoretical results involving discretised automata are available in [161] and [162].

Another branch of algorithms are so called 'estimator' algorithms. Here, an estimate of the penalty probabilities is maintained as the learning proceeds. This added information about the environment is used when updating the action probabilities. The internal state of an automaton is now characterised by the value of the estimate of the penalty probabilities in addition to the action probabilities themselves.

Typically, estimator algorithms yield faster rates of convergence at the cost of additional storage for the penalty probability estimates. Estimator schemes are presented in [163]. Estimator algorithms may also be discretised as described in [164].

More recently, the concept of a multiple response learning automaton has been introduced by Economides [165]. The idea here is to provide different rates of adaptation for different environmental responses. The example Economides gives is for virtual circuit based routing where there are different learning rates depending on the packet delay performance of the selected path. Another type of automaton presented is the 'state dependent linear (SDL)' learning automaton [84]. Now, the reward and penalty parameters can be functions of the network state, where the state could an arbitrary indicator of network performance, not necessarily the same measure as the environment response vector $\underline{\beta}$. The aim with many of these developments is to maintain the excellent steady state performance of traditional algorithms whilst speeding up rate of convergence under changing traffic demands.

A.7. Entropy

Entropy is a measure of the disorganisation of a system and can be mathematically stated as [85]:

$$H = \sum_{i=1}^{n} p_i \log p_i \quad \text{bits} \tag{A.16}$$

where p_i is the probability of performing the i^{th} action. The entropy measure is useful for studying the relative order of the routing scheme and provides an important mechanism for determining when the automata probabilities have converged. For a general network, the total entropy is given by summing the entropy over all automata as follows:

$$H_{total} = \sum_{i \in N} \sum_{j \in D} \sum_{k \in R} p_j^{ik} \log p_j^{ik} \quad \text{bits} \tag{A.17}$$

where N is the set of nodes, D is the set of destinations and R the allowable actions at each automaton in the network [85]. A decrease in the routing scheme entropy can be regarded as a reduction of disorganisation in the system.

Appendix B

Erlang's Formula

The probability that a call requesting use of a line is blocked is given by:

$$B = E(\rho, c) = \frac{\rho^c / c!}{\sum_{k=0}^{c} \rho^k / k!} \tag{B.1}$$

where $\rho = \lambda/\mu$, λ being the call arrival rate, $1/\mu$ the mean call (holding) time and c is the total number of lines available. A minimum bound for blocking probability in a general network for *any* routing mechanism can be calculated by effectively viewing the network as a single link. The minimum network blocking probability assumes that each incoming call only requires one unit (circuit) of resource. Thus, a lower bound on the network-loss probability is given by Erlang's formula:

$$B_{min} = E\left(\sum \rho_{ij}, \sum C_{ij}\right) \tag{B.2}$$

where ρ_{ij} is the offered load (in Erlangs) to the link between nodes i and j, C_{ij} is the capacity (in units/circuits) of the link between nodes i and j, and the summations run over all links ij of the network. In Table B.1, we present the blocking probabilities as a function of the arrival rates using B.2, where each call is assumed to only traverse a single link (2 nodes).

Arrival Rate (calls/min)	Blocking Probability (10-node Network)	Blocking Probability (30-node Network)
5	0	0
10	0	0
15	0	0
20	0	0
25	0	0
30	0	0
35	0	0
40	0	0
45	0.0069	0
50	0.0488	0.0286
55	0.1146	0.101
65	0.2401	0.2341

75	0.3384	0.335
85	0.415	0.413
95	0.476	0.4745

Table B.1 – Erlang Blocking Probabilities for 10 and 30-node Networks.

Appendix C

Traffic Matrices

Here, we provide the traffic matrices for the even and uneven traffics on the 10 node network. Each matrix is specified as a set of probabilities, p_{ij}, giving the probability of node i (row) sending a connection set-up request to node j (column). Thus, for the 10 node network under even traffic demands, each node has a probability of 1/9 of sending a connection request to any other node, apart from itself, this probability being set to 0. The even traffic matrix for the 30 node network will be a 30*30 matrix with all values of 1/9 replaced with 1/29.

10 Node Network, Even Traffic Matrix.

	0	1	2	3	4	5	6	7	8	9
0	0	1/9	1/9	1/9	1/9	1/9	1/9	1/9	1/9	1/9
1	1/9	0	1/9	1/9	1/9	1/9	1/9	1/9	1/9	1/9
2	1/9	1/9	0	1/9	1/9	1/9	1/9	1/9	1/9	1/9
3	1/9	1/9	1/9	0	1/9	1/9	1/9	1/9	1/9	1/9
4	1/9	1/9	1/9	1/9	0	1/9	1/9	1/9	1/9	1/9
5	1/9	1/9	1/9	1/9	1/9	0	1/9	1/9	1/9	1/9
6	1/9	1/9	1/9	1/9	1/9	1/9	0	1/9	1/9	1/9
7	1/9	1/9	1/9	1/9	1/9	1/9	1/9	0	1/9	1/9
8	1/9	1/9	1/9	1/9	1/9	1/9	1/9	1/9	0	1/9
9	1/9	1/9	1/9	1/9	1/9	1/9	1/9	1/9	1/9	0

10 Node Network, Uneven Traffic Matrix.

	0	1	2	3	4	5	6	7	8	9
0	0	1/9	1/9	1/9	1/9	1/9	1/9	1/9	1/9	1/9
1	1/18	0	1/18	1/18	1/18	1/18	1/18	1/18	1/18	5/9
2	1/9	1/9	0	1/9	1/9	1/9	1/9	1/9	1/9	1/9
3	1/18	1/18	1/18	0	1/18	5/9	1/18	1/18	1/18	1/18
4	1/9	1/9	1/9	1/9	0	1/9	1/9	1/9	1/9	1/9
5	1/18	1/18	1/18	5/9	1/18	0	1/18	1/18	1/18	1/18
6	1/18	5/9	1/18	1/18	1/18	1/18	0	1/18	1/18	1/18
7	1/9	1/9	1/9	1/9	1/9	1/9	1/9	0	1/9	1/9
8	1/9	1/9	1/9	1/9	1/9	1/9	1/9	1/9	0	1/9
9	1/18	5/9	1/18	1/18	1/18	1/18	1/18	1/18	1/18	0

Appendix D

Inter-Domain Routing

In Figure D.1, we show how the 30-node network is partitioned for inter-domain (hierarchical) routing experiments. The 30-node network is partitioned into 3 equi-size domains (10 nodes). The border nodes responsible for routing calls to the connected domains for shortest path routing are also marked.

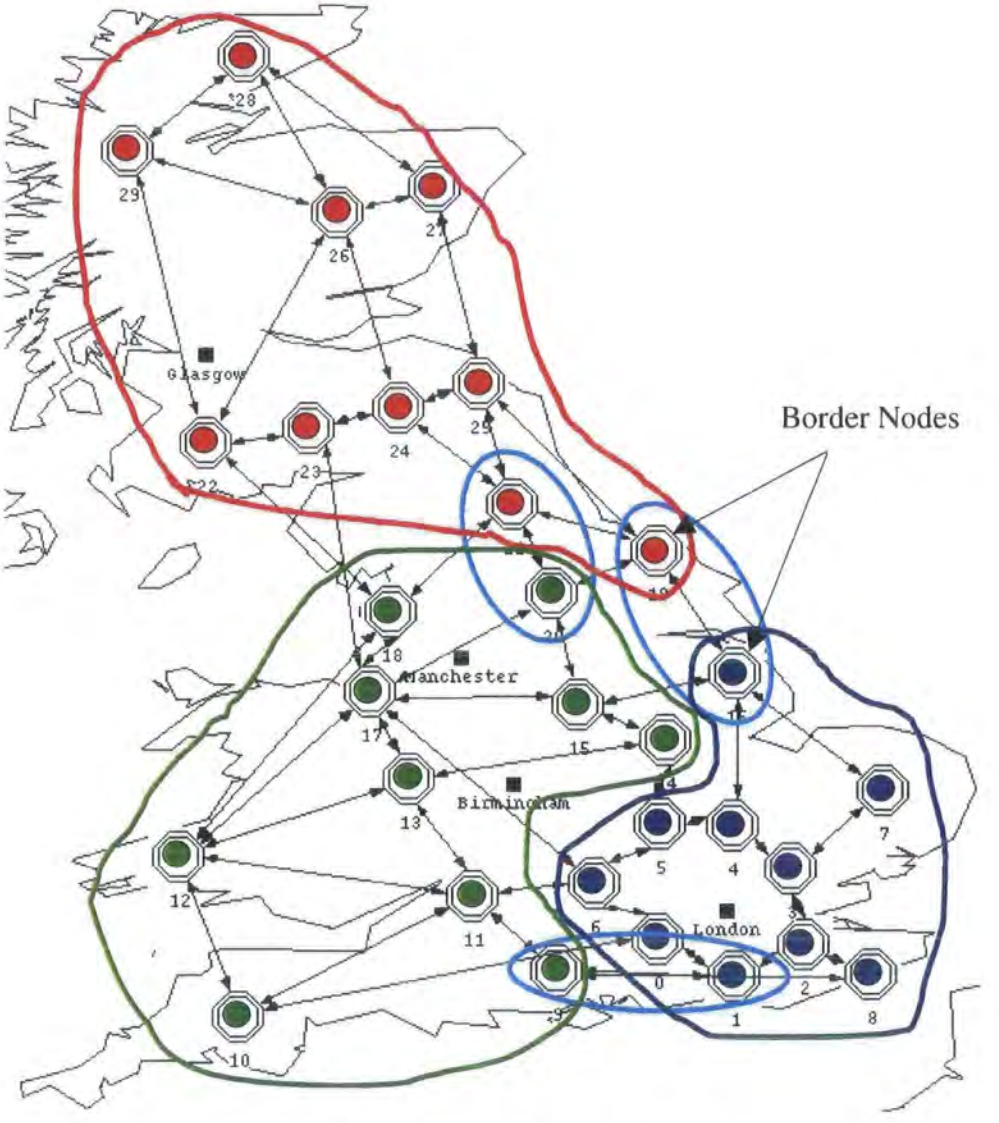


Figure D.1 – Partitioned 30-node Network.

Appendix E

Publications

[1] Reeve, J. M. and Mars, P., 'A review of non-symbolic artificial intelligence techniques for network management and control', *IEE Thirteenth UK Teletraffic Symposium,* Glasgow, March 1996.

[2] Reeve, J. M., Mars, P. and Hodgkinson, T., 'Learning algorithms for multicast routing', *IEE Fifteenth UK Teletraffic Symposium,* Durham, March 1998.

[3] Reeve, J. M., Mars, P. and Hodgkinson, T., 'Learning algorithms for multicast routing in communication networks', *Proceedings of the Tenth Yale Workshop on Adaptive and Learning Systems,* Yale University, June 1998, pp. 70-75.

[4] Reeve, J. M., Mars, P. and Hodgkinson, T., 'Learning algorithms for quality of service multicast routing', *Electronics Letters,* pp. 1195-1197, vol. 34, no. 12, June 1998.

[5] Aranzulla, P., Reeve, J., Mellor, J. and Mars, P., 'Improved stochastic learning automata for routing in ISDNs', *Symposium on Broadband Access Networks, at the European Conference on Networks and Optical Communications 1997 (NOC 97),* Antwerp, Belgium, Ch. 38, pp. 227-231.

[6] Reeve, J. M., Mars, P. and Hodgkinson, T., 'Learning Algorithms for adaptive routing in Integrated-Services networks', submitted to *IEE Proceedings Communications,* Dec. 1997.

[7] Reeve, J. M., Mars, P. and Hodgkinson, T., 'Learning Algorithms for Multicast Routing', submitted to *IEE Proceedings Communications,* Feb/March 1998.

[8] Reeve, J. M., Mars, P. and Hodgkinson, T., 'Learning Algorithms for Minimum Cost, Delay Bounded Multicast Routing in Dynamic Environments.', submitted to *Electronics Letters,* August 1998.

References

[1] Shenker, S., 'Fundamental Design Issues for the Future Internet', *IEEE J. on Selec. Areas in Comms.*, vol. 13, no. 7, Sep. 1995, pp. 1176-1188.

[2] McQuillan, J. M. and Walden, D. C., 'The ARPANET Design Decisions', *Computer Networks*, vol. 1, August 1977.

[3] Clark, D., Shenker, S. and Zhang, L., 'Supporting real-time applications in an integrated services packet network: Architecture and Mechanism', in *Proceedings of ACM Sigcomm*, August 1992, pp. 14-26.

[4] Shenker, S., Clark, D. and Zhang, L., 'A scheduling service model and a scheduling architecture for an Integrated services packet Network', preprint, extended version of [3].

[5] Ferrari, D. and Verma, D. C., 'A scheme for real-time channel establishment in wide-area networks', *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 3, pp. 368-379, 1990.

[6] Demers, A., Kshav, S. and Shenker, S., 'Analysis and Simulation of a Fair Queueing Algorithm', in *Proceedings of ACM Sigcomm*, 1989, pp. 3-12.

[7] Zhang, L., Deering, S., Estrin, D, Shenker, S. and Zappala, D., 'RSVP: A new resource reservation protocol', *IEEE Network Magazine*, vol. 7, no. 5, September 1993, pp. 8-18.

[8] Parekh, A. K. and Gallager, R. G., 'A generalised processor sharing approach to flow control in integrated services networks: The single node case.', *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, June 1993, pp. 344-357.

[9] Parekh, A. K. and Gallager, R. G., 'A generalised processor sharing approach to flow control in integrated services networks: The multiple node case', *IEEE/ACM Transactions on Networking*, vol. 2, no. 2, April 1994, pp. 137-150.

[10] Hyman, J., Lazar, A. A. and Giovannia, P., 'Real-time scheduling with quality of service constraints', *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 7, September 1991, pp. 1052-1063.

[11] Jamin, S., Danzig, P., Shenker, S. and Zhang, L., 'A measurement based admission control algorithm for integrated services packet networks', In *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, pp. 56-70, 1997.

[12] Floyd, S. and Jacobson, V., 'Link-Sharing and Resource Management Models for Packet Networks', *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, August 1995, pp. 365-386.

[13] Wroclawski, J., 'Specification of the Controlled-Load Network Element Service', RFC 2211, September 1997.

[14] Shenker, S., Partridge, C. and Guerin, R., 'Specification of Guaranteed Quality of Service', RFC 2212, September 1997.

[15] Nichols, K., Jacobson, V. and Zhang, L., 'A Two-bit Differentiated Services Architecture for the Internet', Internet Draft, draft-nichols-diff-svc-arch-00.txt, November 1997.

[16] Lefelhocz, C. L., Lyles, B., Shenker, S. and Zhang, L., 'Congestion Control for Best-Effort Service: Why We Need a New Paradigm', *IEEE Network Magazine*, Jan/Feb 1996, pp. 10-19.

[17] Braden, R., Clark, D. and Shenker, S., 'Integrated Services in the Internet Architecture: an Overview', RFC 1633, June 1994.

[18] Hawker, I. and Cochrane, P., 'The 'Really Intelligent Network'', *British Telecommunications Engineering*, vol. 13, Jan. 1995, pp. 326-334.

[19] Baransel, C., Dobosiewicz, W. and Gburzynski, P., 'Routing in Multihop Packet Switching Networks: Gb/s Challenge', *IEEE Network Magazine*, May/June 1995, pp. 38-61.

[20] Partridge, C. (ed), 'Workshop Report: Internet research steering group workshop on very high speed networks', RFC 1152, April 1990.

[21] Steenstrup, M., 'Routing in Communications Networks', Prentice Hall, 1995.

[22] Harvey, J. J., 'Expert Systems : An Introduction', *Electrical Communication*, vol. 60, no. 2, 1986, pp. 100-108.

[23] Goyal, S. K. and Worrest, R. W., 'Expert System Applications To Network Management', Expert System Applications to Telecommunications, Wiley Series in Telecommunications, 1988, pp. 3-44.

[24] Hariri, S. and Jabbour, K., 'An Expert System for Network Management', *Tenth Annual International Pheonix Conference on Computers and Communications*, 1991, Ch. 119, pp. 580-586.

[25] Funabashi, M., Maeda, A., Moroola, Y., Mori, K. and Works, O., 'Fuzzy and Neural Hybrid Expert Systems: Synergetic AI', *IEEE Expert*, August 1995, pp. 32-40.

[26] Haykin, S., 'Neural Networks – A Comprehensive Foundation', Macmillan, 1994.

[27] Hall, C. and Smith, R., 'Pitfalls in the application of neural networks for process control', Neural Networks for Control and Systems, Eds. K. Warwick, G. W. Irwin and K. J. Hunt, IEE Control Engineering Series, 1992, pp. 243-256.

[28] ATM Forum Technical Committee, 'Traffic Management Specification, Version 4.0', af-tm-0056.000, April 1996.

[29] Habib, I. W. and Saadawi, T. N., 'Controlling Flow and Avoiding Congestion in Broadband Networks', *IEEE Comms. Mag.*, vol. 29, no. 10, Oct. 1991, pp. 46-53.

[30] Takahashi, T. and Hiramatsu, A., 'Integrated ATM Traffic Control by Distributed Neural Networks', Proc. *ISS '90 Stockholm*, May 1990, pp. 59-65.

[31] Hiramatsu, A., 'ATM Communications Network Control and Link Capacity Control by Distributed Neural Networks', *IEEE Trans. Neural Networks*, vol. 1, no. 1, March 1990, pp. 122-130.

[32] Hiramatsu, A., 'Training Techniques for Neural Network Applications in ATM', *IEEE Comms. Mag.*, Oct 1995, pp. 58-67.

[33] Jamin, S., Shenker, S. J. and Danzig, P. B., 'Comparison of Measurement-based Admission

Control Algorithms for Controlled-Load Service', in *IEEE Infocom 1997*, pp. 973-980.

[34] Hiramatsu, A., 'Integration of ATM Call Admission Control and Link Capacity Control by Distributed Neural Networks', *IEEE J. Selec. Areas in Comms.*, vol. 9, no. 7, Sep. 1991, pp. 1131-1138.

[35] Morris, R. J. T. and Samadi, B., 'Neural Network Control of Communication Systems', *IEEE Trans. on Neural Networks*, vol. 5, no. 4, July 1994, pp. 639-650.

[36] Nordstrom, E., Carlstrom, J., Gallmo, O. and Asplund, L., 'Neural Networks for Adaptive Traffic Control in ATM Networks', *IEEE Comms. Mag.*, Oct. 1995, pp. 43-49.

[37] Gallmo, O. and Asplund, L., 'Reinforcement Learning by Construction of Hypothetical Targets', in Proc. Of the *Int'l Workshop on Applications of Neural Networks to Telecommunications* 2 (IWANNT-95), Stockholm, Sweden, 1995, pp. 300-307.

[38] Fan, Z. and Mars, P., 'Performance analysis of an ATM cell multiplexer with MMPP input and a neural connection admission approach', *Int'l Conference on Communication Technology (ICCT'96)*, Bejing, China, May 1996, pp. 916-919.

[39] Fan, Z. and Mars, P., 'Application of artificial neural networks to effective bandwidth estimation in ATM networks', in *IEEE Int'l Conf. On Neural Networks (ICNN'96)*, Washington DC, USA, June 1996, pp. 1951-1956.

[40] Brandt, H. et al., 'A Hybrid Neural Network Approach to ATM Admission Control', Proc. Of the *Int'l Switching Symp. (ISS'95)*, p. P.b6, Berlin, April 1995.

[41] Fan, Z. and Mars, P., 'A congestion controller for ATM networks using reinforcement learning', *9th Yale Workshop on Adaptive and Learning Systems*, Yale University, CT, USA, June 1996, pp. 77-82.

[42] Chen, X. and Leslie, I. M., 'Neural adaptive congestion control for broadband ATM networks', *IEE Proceedings on Communications*, vol. 139, no. 2, June 1992, pp. 233-240.

[43] Yang, C. Q. and Reddy, A. V. S., 'A taxonomy for congestion control algorithms in packet switching networks', *IEEE Network*, vol. 9, no. 4, 1995, pp. 34-45.

[44] Hall, J. and Mars, P., 'The Limitations of Artificial Neural Networks for Traffic Prediction', *Third IEEE Symposium on Computers and Communications*, Athens, Greece, 1998, pp. 8-12.

[45] Fan, Z. and Mars, P., 'ATM traffic prediction using FIR neural networks', *3rd IFIP workshop on performance modelling and evaluation of ATM networks*, Ilkley, UK, July 1995.

[46] Fan, Z. and Mars, P., 'An access flow control scheme for ATM networks using neural-network-based traffic prediction', *IEE Proceedings on Communications*, vol. 144, no. 5, 1997, pp. 295-300.

[47] Tarraf, A., Habib, I. And Saadawi, T., 'A Novel Neural Network Traffic Enforcement Mechanism for ATM Networks', *IEEE J. on Selec. Areas of Comms.*, vol. 12, no. 6, August 1994, pp. 1088-1096.

[48] Marrakchi, A. and Troudet, T., 'A Neural Network Arbitrator for Large Crossbar Packet Switches', *IEEE Trans. on Circuits and Systems*, vol. 36, no. 7, July 1989, pp. 1039-1041.

[49] Park, Y. and Lee, G., 'Applications of Neural Networks in High-Speed Communication Networks', *IEEE Comms. Mag.*, Oct 1995, pp. 68-74.

[50] Rauch, H. E. and Winarske, T., 'Neural Networks for Routing Communication Traffic', *IEEE Control Systems Magazine*, April 1988, pp. 26-30.

[51] Wang, C. and Weissler, P. N., 'The Use of Artificial Neural Networks for Optimal Message Routing', *IEEE Network*, March/April 1995, pp. 16-24.

[52] Parisini,T. and Zoppoli, R., 'Team theory and neural networks for dynamic routing in traffic and communication networks', *INFORMATION AND DECISION TECHNOLOGIES*, 1993, Vol.19, No.1, pp.1-18.

[53] Gelenbe, E., Ghanwani, A. and Srinivasan, V., 'Improved Neural Heuristics for Multicast Routing', *IEEE J. on Selec. Areas in Comms.*, vol. 15, no. 2, Feb. 1997, pp. 147-155.

[54] Zadeh, L. A., 'Fuzzy Sets', Information and Control, 8:330-353, 1965.

[55] Harris, C. J., et al (Eds), 'Intelligent control aspects of fuzzy logic and neural nets', World Scientific, 1993.

[56] Uehara, K. and Hirota, K., 'Fuzzy Connection-Admission-Control for ATM networks based on Possibility Distribution of Cell Loss Ratio', *IEEE JSAC*, vol. 15, no. 2, February 1997, pp. 179-190.

[57] Cheng, R. G. and Chang, C. J., 'Design of a Fuzzy Traffic Controller for ATM networks', *IEEE/ACM Trans. Networking*, vol. 4, no. 3, June 1996, pp. 460-469.

[58] Catania, V., Ficili, G., Palazzo, S. and Panno, D., 'A Comparative Analysis of Fuzzy Versus Conventional Policing Mechanisms for ATM Networks', *IEEE/ACM Trans. On Networking*, vol. 4, no. 3, June 1996, pp. 449-459.

[59] Ndousse, T. D., 'Fuzzy Neural Control of Voice Cells in ATM Networks', *IEEE. J. Selec. Areas in Comms.*, vol. 12, no. 9, Dec. 1994, pp. 1488-1494.

[60] Bonde, A. R. and Ghosh, S., 'A Comparative Study of Fuzzy Versus 'Fixed' Thresholds for Robust Queue Management in Cell-Switching Networks', *IEEE/ACM Trans. Networking*, vol. 2, no. 4, Aug. 1994, pp. 337-344.

[61] Chemouil, P., Khalfet, J. and Lebourges, M., 'A Fuzzy Control Approach for Adaptive Traffic Routing', *IEEE Comms. Mag.*, July 1995, pp. 70-76.

[62] Tanaka, Y., Miyakoshi, K. and Akiyama, M., 'Dynamic Routing by the Use of Hierarchical Fuzzy System', *IEICE Trans. Communications*, Vol. 74, no. 12, 1991, pp. 4000-4006.

[63] Nwana, H.S. : 'Software Agents: An Overview', *Knowledge Engineering Review*, Vol. 11, No. 3, pp. 205-244, October/November 1996.

[64] Appleby, S. and Steward, S. : 'Mobile Software Agents for Control in Telecommunications Networks', *BT Technological Journal* 12 (2), pp. 104-113, April 1994.

[65] Brooks, R.A. : 'A Robust Layered Control System for a Mobile Robot', *IEEE Journal of Robotics and Automation*, 2 (1), pp. 14-23, 1986.

[66] Schoonderwoerd, R., Holland, O. and Bruten, J., 'Ant-like agents for load balancing in telecommunications networks', *In Proceedings of the First International Conference on Autonomous Agents*, pp. 209-216, ACM Press.

[67] Caro, G. D. and Dorigo, M., 'AntNet: A Mobile Agents Approach to Adaptive Routing', *Proceedings of the 31st Hawaii International Conference on Systems*, Big Island of Hawaii, Jan '98.

[68] Moy, J., 'OSPF Version 2', RFC 1247, July 1991.

[69] Legedza, U., Wetherall, D. and Guttag, J., 'Improving the Performance of Distributed Applications Using Active Networks', in Proc. of *IEEE Infocom* 1998, pp. 590-599.

[70] Muller, J.P., Pishel, M. and Thiel, M. : 'Modelling Reactive Behaviour in Vertically Layered Agent Architectures', In Wooldridge, M. and Jennings, N. (eds.), *Intelligent Agents,* Lecture Notes in Artificial Intelligence 890, Heidelberg: Springer Verlag, pp. 261-276, 1995.

[71] Ferguson, I.A. : 'Touring Machines: An Architecture for Dynamic, Rational, Mobile Agents', PhD Thesis, Computer Laboratory, University of Cambridge, UK, 1992.

[72] Narendra, K. and Thathachar, M., 'Learning Automata – An introduction', Prentice-Hall, 1989.

[73] Mason, L. G. and Gu, X. D., 'Learning Automata Models for Adaptive Flow Control in Packet-Switching Networks', in *Adaptive and Learning Systems,* K. S. Narendra (Ed.), New York: Plenum Press, 1986, pp. 213-228.

[74] Meybodi, M. R. and Lakshmivarahan, S., 'A Learning Approach to Priority Assignment in a Two Class M/M/1 Queuing System with Unknown Parameters', Proc. *Third Yale Workshop on Applications of Adaptive Systems Theory*, Yale University, 1982, pp. 106-109.

[75] El-Fattah, Y. M., Boyer, P., Dupuis, A. and Romoeuf, L., 'Use of a Learning Automaton for Control of Service Activity', Proc. *Fourth Yale Workshop on Applications of Adaptive Systems Theory*, Yale University, 1985, pp. 124-129.

[76] Hall, J. and Mars, P., 'Satisfying QoS with a Learning Based Scheduling Algorithm', in 1998 *Sixth International Workshop on Quality of Service (IWQOS)*, Nappa, pp. 171-173.

[77] Akselrod, B. and Langholz, G., 'A simulation study of advanced routing methods in a multipriority telephone network', *IEEE Trans. on Systems, Man and Cybernetics*, vol. 15, no. 6, pp. 730-736, 1985.

[78] Narendra, K. S., Wright, E. A. and Mason, L. G., 'Application of learning automata to telephone traffic routing and control', *IEEE Trans. on Sys., Man and Cyb.*, vol. 7, no. 11, pp. 785-792, 1977.

[79] Narendra, K. S. and Thathachar, M. A. L., ' On the behaviour of a learning automaton in a changing environment with application to telephone traffic routing', *IEEE Trans on Sys., Man and Cyb.*, vol. 10, no. 5, pp. 262-269, 1980.

[80] Zgierski, J. R. and Oommen, B. J., 'SEAT: An object-oriented simulation environment using learning automata for telephone traffic routing', *IEEE Trans. on Sys., Man and Cyb.*, vol. 24, no. 2, pp. 349-356, 1994.

[81] Narendra, K. S. and Mars, P., 'The use of learning algorithms in telephone traffic routing – a

methodology', *Automatica*, vol. 19, no. 5, pp. 495-502, 1983.

[82] Eshragh, N., 'Dynamic routing in circuit-switched non-hierarchical networks', PhD Thesis, University of Durham, 1989.

[83] Economides, A. A.m Ioannou, P. A. and Silvester, J. A., 'Decentralized adaptive routing for virtual circuit networks using stochastic learning automata', Proc. *IEEE Infocom 88* Conference, 1988, pp. 613-622.

[84] Economides, A. A., 'Learning automata routing in connection-oriented networks', *Int. Journal of Communication Systems*, vol. 8, pp. 225-237, 1995.

[85] Chrystall, M. S., 'Adaptive control of communication networks using learning automata', PhD Thesis, Robert Gordon's Institute of Technology, Aberdeen, 1982.

[86] Vasilakos, A. V., 'Learning automata for data communication routing problem', *Kybernetika*, vol. 25, no. 6, pp. 486-493, 1989.

[87] Vasilakos, A. V. and Koubias, S. A., 'On routing and performance comparison of techniques for packet-switched networks using learning automata', *IEEE Infocom 1988*, pp. 109-113.

[88] Vasilakos, A. V. and Paximadis, C. T., 'Fault-Tolerant Routing Algorithms Using Estimator Discretized Learning Automata for High-Speed Packet-Switched Networks', *IEEE. Trans. on Reliability*, vol. 43, no. 4, Dec. 1994.

[89] Nedzelnitsky, O. V. and Narendra, K. S., 'Nonstationary models of learning automata routing in data communication networks', *IEEE Trans. on Sys., Man and Cyb.*, vol. 17, no. 6, pp. 1004-1015, 1987.

[90] Bertsekas, D. and Gallager, R., 'Data Networks', Prentice-Hall International, 1987.

[91] Lin, W. and Kumar, P. R., 'Optimal Control of a Queueing System with Two Heterogeneous Servers', *IEEE Transactions on Automatic Control*, vol. AC-29, no. 8, August 1984. pp. 696-703.

[92] Hedrick, C., 'Routing Information Protocol', RFC 1058, June 1988.

[93] Crawley, E., Nair, R., Rajagopalan, B. and Sandick, H., 'A Framework for QoS-based Routing in the Internet', draft-ietf-qosr-framework-01.txt, July 1997.

[94] Zappala, D. and Estrin, D., 'Alternate Path Routing and Pinning for Interdomain Multicast Routing', University of Southern California (USC) Computer Science Technical Report #97-655.

[95] Guerin, R., Orda, A. and Williams, D., 'QoS Routing Mechanisms and OSPF Extensions', IETF Internet Draft, draft-guerin-qos-routing-ospf-00.txt, November 1996 .

[96] Wang, Z. and Crowcroft, J., 'Quality-of-Service Routing for Supporting Multimedia Applications', *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, September 1996, pp. 1288-1234.

[97] Ma, Q., Steenkiste, P. and Zhang, H., 'Routing High-bandwidth Traffic in Max-min Fair Share Networks', in *ACM SIGCOMM96*, pp. 206-217, Stanford, August 1996.

[98] Ma, Q. and Steenkiste, P., 'Quality-of-Service Routing for Traffic with Performance Guarantees', available from http://www.cs.cmu.edu/qma, 1997.

[99] Salama, H. F., Reeves, D. S. and Viniotis, Y., 'A Distributed Algorithm for Delay-Constrained Unicast Routing', in Proceedings of *IEEE INFOCOM '97*, pp. 92-100, Kobe, Japan, April 1997.

[100] Breslau, L., 'Adaptive Source Routing of Real-Time Traffic in Integrated Services Networks', PhD Thesis, University of Southern California (USC), December 1995.

[101] Lee, W. C., Hluchi, M. G. and Humblet, P. A., 'Routing Subject to Quality of Service Constraints in Integrated Communication Networks', *IEEE Network Magazine*, pp. 46-55, July/August 1995.

[102] Hurley, B. R., Seidl, C. J. R. and Sewell, W. F., 'A Survey of Dynamic Routing Methods for Circuit-Switched Traffic', *IEEE Communication Magazine*, Vol. 25, No. 9, September 1987, pp. 13-21.

[103] Steenstrup, M., 'Routing in Communications Networks', Prentice Hall, 1995, Chapter 1, 'Dynamic Alternative Routing'.

[104] Ash, G. R., Cardwell, R. H. and Murray, R. P., 'Design and Optimisation of Networks with Dynamic Routing', The *Bell System Technical Journal*, Vol. 60, No. 8, October 1981, pp. 1787-1820.

[105] Paxson, V, 'End-to-end routing behaviour in the Internet', *IEEE/ACM Trans. On Networking*, vol. 5, no. 5, 1997, pp. 601-615.

[106] Yum, T. P. and Shwartz, M., 'Comparison of Routing Procedures for Circuit-Switched Traffic in Non-hierarchical Networks', *IEEE Trans. Communications*, pp. 534-544, May 1987.

[107] ATM Forum PNNI subworking group, 'Private Network-Network Interface Specification v1.0 (PNNI 1.0)', afpnni-0055.00, March 1996.

[108] Shaikh, A., Rexford, J. and Shin, K. G., 'Dynamics of Quality-of-Service Routing with Inaccurate Link-State Information' from http://www.eecs.umich.edu/~ashaikh/work/index.html

[109] Shaikh, A., Rexford, J. and Shin, K. G., 'Efficient Precomputation of Quality-of-Service Routes', from http://www.eecs.umich.edu/~ashaikh/work/index.html

[110] Guerin, R. and Orda, A., 'QoS-based Routing in Networks with Inaccurate Information: Theory and Algorithms', IBM Research Report, RC 20515, July 1996. Conference version appeared in the Proceedings of *IEEE INFOCOM '97*, Kobe, Japan, pp. 75-83.

[111] Lorenz, D. H. and Orda, A., 'QoS Routing in Networks with Uncertain Parameters', in Proceedings of *IEEE INFOCOM '98*, pp. 3-10, San Francisco, April 1998.

[112] Matta, I. and Udaya Shankar, A., 'Type-of-Service Routing in Datagram Delivery Systems', *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, October 1995, pp. 1411-1425.

[113] Salama, H., F., Reeves, D., S. and Viniotis, Y., 'Evaluation of Multicast Routing Algorithms for Real-Time Communication on High-Speed Networks', *IEEE Journal on Selected Areas in Communications*, pp. 332-345, vol. 15, no. 3, April 1997.

[114] Nagrath, I. J. and Gopal, M., 'Control Systems Engineering', John Wiley and Sons, 2nd Edition, 1986.

[115] Calvert, K. L., Doar, M. B. and Zegura, E. W., 'Modeling Internet Topology', *IEEE Comms. Mag.*,

vol. 35, no. 6, 1997, pp. 160-163.

[116] Jain, R., 'The art of computer systems performance analysis – Techniques for Experimental Design, Measurement, Simulation, and Modelling', Jon Wiley and Sons, 1991.

[117] Rudin, H., On Routing and 'Delta Routing': A Taxonomy and Performance Comparison of Techniques for Packet Switched Networks', *IEEE Transactions on Communications*, January 1976, pp. 43-59.

[118] Johnson, D., Brown, G. N., Botham, C. P., Beggs, S. L. and Hawker, I., 'Distributed Restoration in Telecommunication Networks', *BT Technology Journal*, vol. 12, no. 2, April 1994, pp. 67-76.

[119] Guerin, R., Ahmadi, H. and Naghshineh, M., 'Equivalent capacity and its application to bandwidth allocation in high-speed networks', *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 9, September 1991, pp. 968-981.

[120] Postel, J., 'Internet Protocol', RFC 791, September 1981.

[121] Estrin, D., Zappala, D., Li, T., Rekhter, Y. and Varadhan, K., 'Source Demand Routing: Packet Format and Forwarding Specification (Version 1)', RFC 1940, May 1996.

[122] Kelly, F. P., 'Network Routing', *Phil. Trans. R. Soc. Lond. A*, 1991, pp. 343-367.

[123] Mitra, D., Gibbens, R. J. and Huang, B. D., 'Analysis and Optimal Design of Aggregated-Least-Busy-Alternative Routing on Symmetric Loss Networks with Trunk Reservations', Proceedings of the *Thirteenth International Teletraffic Congress*, Copenhagen, Amsterdam, North-Holland, 1991.

[124] Gazdzicki, P., Lambadaris, I. and Mazumdar, R., 'Blocking probabilities for large multirate erlang loss systems', Adv. Appl. Prob., 1993, vol. 25, pp. 997-1009.

[125] Kaufman, J.S., 'Blocking in a Shared Resource Environment', *IEEE Transactions on Communications*, vol. com-29, no. 10, October 1981, pp. 1474-1481.

[126] Loughheed, K. and Rekhter, Y., 'Border Gateway Protocol', RFC 1105, June 1989.

[127] Castineyra, I., Chiappa, N. and Steenstrup, M., 'The Nimrod Routing Architecture', RFC 1992, August 1996.

[128] Khanna, A. and Zinky, J., 'The Revised ARPANET Routing Metric', *ACM Sigcomm Symposium on Communications Architectures and Protocols*, 1989, vol. 4, Ch, 29, no. 4, pp. 45-56.

[129] McQuillan, J. M., Richer, I. And Rosen, E., 'The new routing algorithm for the ARPANET', *IEEE Trans. On Comms.*, vol. 25, no. 5, pp. 711-719, May 1980.

[130] Bertsekas, D. P., 'Dynamic behaviour of shortest path routing algorithms for communication networks', *IEEE Trans. On Automatic Control*, vol. 27, no. 1, Feb. 1982, pp. 60-74.

[131] Postel, J., 'Transmission Control Protocol (TCP)', RFC 793, September 1981.

[132] Postel, J., 'Internet Control Message Protocol (ICMP)', RFC792, September 1981.

[133] Danzig, P. and Jamin, S., 'tcplib: A library of TCP internetwork traffic characteristics', Comp. Sci. Dept., Univ. Southern California, Rep. CS-SYS-91-01, 1991. Available via FTP to catarina.usc.edu as *pub/jamin/tcplib/tcplib.tar.Z.*

[134] Paxson, V. and Floyd, S., 'Wide Area Traffic: The failure of Poisson modelling', *IEEE/ACM*

Trans. On Networking, vol. 3, no. 3, June 1995, pp. 226-244.

[135] Deering, S., 'Multicast Routing in a Datagram Internetwork', PhD thesis, Stanford University, December 1991.

[136] Pusateri, T., 'Distance Vector Multicast Routing Protocol', Internet draft, draft-ietf-idmr dvmrp-v3-06.txt, March 1998.

[137] Moy, J., 'Multicast Extension to OSPF', RFC 1584, May 1994.

[138] Deering, S. E. *et al,* 'The PIM architecture for wide-area multicast routing', *IEEE/ACM Trans. Networking*, vol. 4, no. 2, pp. 153-162, April 1996.

[139] Pasquale, J. C., Polyzos, G. C. and Xylomenos, X., 'The multimedia multicasting problem', available from http://www-csl.ucsd.edu/. Also in, *Multimedia Systems*, 1998, Vol.6, No.1, pp. 43-59.

[140] Winter, P., 'Steiner problem in networks: A survey', *IEEE Networks*, vol. 17, no. 2, pp. 129-167, 1987.

[141] Salama, H. F., 'Multicast Routing for Real-Time Communication on High-Speed Networks', PhD thesis, North Carolina State University, 1996.

[142] Kou, L., Markowsky, G. and Berman, L., 'A Fast Algorithm for Steiner Trees', *Acta Informatica*, vol. 15, pp. 141-145, 1981.

[143] Doar, M. and Leslie, I., 'How bad is naïve multicast routing?', Proceedings of *IEEE Infocom '93*, pp. 82-89.

[144] Ballardie, A. J., 'A new approach to multicast communication in a datagram internetwork', PhD dissertation, University College London, 1995.

[145] Wei, L. and Estrin, D., 'The trade-offs of multicast trees and algorithms', in Proceedings of *the 4th International Conference on Computer Communications and Networks,* September 1994.

[146] Thaler, D. G. and Chinya, V. R., 'Distributed Center-Location Algorithms', *IEEE Journal on selected areas in communications*, vol. 15, no. 3, 1997.

[147] Calvert, K. L., Zegura, E. W. and Donahoo, M. J., 'Core Selection Methods for Multicast Routing', in Proc. *ICCCN 1995*, Las Vegas, Nevada.

[148] Wall, D. W., 'Mechanisms for broadcast and selective broadcast', PhD thesis, Stanford University, June 1980.

[149] Handley, M., Crowcroft, J. and Wakeman, I., 'Hierarchical protocol independent multicast (HPIM)', University College London, November 1995.

[150] Donahoo, M. J. and Zegura, E. W., 'Core Migration for Dynamic Multicast Routing', in Proc. *ICCCN* 1996, Washington DC.

[151] Waxman, B. M., 'Routing of Multipoint Connections', *IEEE Journal on Selected Areas in Communications*, pp. 1617-1622, vol. 6, no. 9, December 1988.

[152] Ballardie, A., 'Core Based Trees (CBT) Multicast Routing Architecture', RFC 2201, September 1997.

[153] Salama, H., 'MCRSIM© Second Edition', September 1997, available from ftp://ftp.csc.ncsu.edu/pub/rtcomm/mcrsim.html.

[154] KMB code from Michael J. Alexander, Assistant Professor, School of Electrical Engineering/Computer Science, Washington State University, see http://www.eecs.wsu.edu/~alexande.

[155] Rouskas, N. G. and Baldine, I., 'Multicast Routing with End-to-End Delay and Delay Variation Constraints', *IEEE Journal on Selected Areas in Communications*, pp. 346-356, vol. 15, no. 3, April 1997.

[156] Hong, S., Lee, H. and Hwan Park, B., 'An Efficient Multicast Routing Algorithm for Delay-Sensitive Applications with Dynamic Membership', in Proceedings of *IEEE INFOCOM '98*, pp. 1433-1440.

[157] Kompella, V. P., Pasquale, J. C. and Polyzos, G. C., 'Multicast routing for multimedia communication', *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 286-292, June 1993.

[158] Zhu, Q., Parsa, M. and Garcia-Luna-Aceves, J. J., 'A source-based algorithm for near-optimum delay-constrained multicasting', in Proceedings of *IEEE Infocom '95*, March 1995, pp. 377-385.

[159] Zhang, Z., Sanchez, C., Salkewicz, B. and Crawley, E., 'QoS Extensions to OSPF', Internet Draft, draft-zhang-qos-ospf-01.txt, September, 1997.

[160] Philip Aranzulla, formerly with University of Durham, now at Cable and Wireless, private communication, 1998.

[161] Oommen, B. J. and Christensen, J. P. R., 'e-optimal discretised linear reward-penalty learning automata', *IEEE Transactions on Systems, Man and Cybernetics,* vol. 18, no. 3, pp. 451-458, May/June 1988.

[162] Oommen, B. J. and Hansen, E. R., 'The asymptotic optimality of discretised linear reward-inaction learning automata', *IEEE Transactions on Systems, Man and Cybernetics,* vol. 14, no. 3, pp. 542-545, May/June 1984.

[163] Thathatchar, M. A. L. and Sastry, P. S., 'A new approach to the design of reinforcement schemes for learning automata', *IEEE Trans. on Sys., Man and Cyb.*, vol. 15, no. 1, pp. 168-175, Jan/Feb 1985.

[164] Oommen, J. B. and Lanctot, K. J., 'Discretised pursuit learning automata', *IEEE Trans. on Sys., Man and Cyb.*, vol. 20, no. 4, pp. 931-938, July/August 1990.

[165] Economides, A. A., 'Multiple response learning automata', *IEEE Trans. on Sys., Man and Cyb, part-B,* 1996, vol. 26, no. 1, pp. 153-156.